C - INTRODUCTION TO CP/M ON NORTH STAR DISK

THE PURPOSE OF THIS SECTION IS TO EXPLAIN HOW TO GET YOUR CP/M ON NORTH STAR SYSTEM "ON THE AIR".

THE FOLLOWING IS VERY IMPORTANT. BEFORE MAKING ANY ATTEMPT TO USE YOUR NEW SYSTEM, READ YOUR MANUALS. THEN, RE-READ THEM! TRY TO GET AN OVERALL FEEL FOR WHAT CP/M IS ABOUT. TIME SPENT NOW WILL LEAD TO MUCH LESS LATER FRUSTRATION. THE BOOKLET ENTITLED "INTRODUCTION TO CP/M FEATURES AND FACILITIES" IS THE MOST IMPORTANT AT THIS STAGE.

WRITE PROTECT THE DISK SUPPLIED AND MAKE A COPY OF IT USING NORTH STAR DOS FACILITIES. NEVER WRITE ON THE ORIGINAL DISK SO THAT YOU WILL ALWAYS HAVE A CORRECT ORIGINAL COPY OF THE SYSTEM, EVEN IF EVERYTHING ELSE BLOWS UP.

THE NORMAL OPERATION OF THIS SYSTEM IS TO USE THE PROM ON THE NORTH STAR CONTROLLER BOARD TO BOOT IN A SINGLE SECTOR (SECTOR 4). CONTROL IS THEN AUTOMATICALLY TRANSFERRED TO THE PROGRAM THAT WAS BOOTED IN, WHICH LOADS CP/M FROM THE FIRST THREE TRACKS ON THE DISK TO THE RAM LOCATION OF CP/M (2900H TO 4EFFH IN A 16K SYSTEM). AFTER LOADING, CONTROL TRANSFERS TO CP/M WHICH WILL SIGN ON WITH A MESSAGE, IF THE CONSOLE I/O PATCHES ARE AREADY CORRECT FOR YOUR INTERFACE HARDWARE.

THE SYSTEM ALREADY CONTAINS A TELETYPE DRIVER WHICH IS THE SAME AS LISTED IN SECTION H OF THIS MANUAL. THIS DRIVER IS SUPPLIED AS AN EXAMPLE, ALTHOUGH IT IS A VERY COMMONLY USED ROUTINE IN ITSELF, CORRESPONDING TO THE ALTAIR SIO (REV NON-ZERO). IF YOUR I/O SYSTEM HAPPENS TO BE THE SAME, YOU ARE LUCKY AND NO PATCH NEED BE MADE. HOWEVER, IN MOST CASES A PATCH WILL BE NEEDED FOR YOUR OWN SPECIFIC I/O - NOW REFER TO THE SECTION ON "PATCHING THE USER AREA" AND DO THIS. REFER TO THE DOCUMENTATION ACCOMPANYING YOUR I/O HARDWARE FOR THE NECESSARY ROUTINES TO SUPPLY.

AT THIS POINT YOU SHOULD HAVE COMPLETED THE FOLLOWING STEPS:

1) REVIEWED THE DOCUMENTATION DESCRIBING CP/M FEATURES
2) BACKED-UP YOUR SUPPLIED SYSTEM DISKETTE AND FILES
3) PATCHED IN YOUR CUSTOM CONSOLE I/O ROUTINES

ONE LAST PATCH MAY BE NECESSARY IF YOU HAVE A NON-STANDARD PROM ON YOUR NORTH STAR CONTROLLER BOARD. THE STANDARD PROM RESIDES AT LOCATION E900H AND OPERATES BY BOOTING IN SECTOR FOUR OF THE DISK TO RAM LOCATION 2000H, WHICH IS THE BEGINNING POINT OF THE NORTH STAR DOS. IF YOUR PROM IS NOT AT E900H OR IF YOUR DOS IS NOT AT 2000H, THEN YOUR PROM IS NON-STANDARD AND YOU MUST REFER TO THE SECTION

ON "PATCHING A NON-STANDARD PROM". IF YOUR PROM IS IN FACT STANDARD, THEN YOU HAVE NO PATCH TO MAKE.

ONCE YOU HAVE MADE THE USER AREA PATCH FOR CONSOLE I/O AND THE NON-STANDARD PROM PATCH IF NECESSARY, YOU ARE READY TO TRY OUT THE SYSTEM. TO DO THIS, PUT IN YOUR PATCHED DISK AND BOOT IN THE SYSTEM IN EXACTLY THE SAME WAY THAT YOU BOOT IN THE NORTH STAR DOS. ONE EASY WAY TO DO THIS IS TO FIRST BOOT IN THE NORTH STAR DOS FROM A REGULAR NORTH STAR DISK, THEN USE THE DOS COMMAND:

```
JP E900 CR
```

IF YOU HAVE BEEN SUCESSFUL IN YOUR PATCHES, THE SYSTEM WILL SIGN ON WITH THE MESSAGE:

```
CP/M ON NORTH STAR DISK
16K VERSION 1.41
COPYRIGHT (C) 1978 LIFEBOAT ASSOCIATES
A>
```

WHEN YOU GET THIS FAR YOU ARE UP AND ON THE AIR.

FIRST EXPERIMENT WITH THE BUILT-IN COMMANDS DESCRIBED IN SECTION 2.1 AND ONWARDS IN THE "INTRODUCTION TO CP/M FEATURES AND FACILITIES". AS DESCRIBED, THESE FIVE FUNTIONS ARE ERA, DIR, REN, SAVE AND TYPE.

THE NEXT AREA OF EXPLORATION WILL BE IN THE TRANSIENT COMMANDS, MENTIONED IN SECTION 6 OF "INTRODUCTION TO ...". IT SHOULD BE NOTED THAT AS PURCHASERS OF CP/M ON NORTH STAR DISK, YOU HAVE RECEIVED SPECIAL VERSIONS OF TWO OF THESE: NSYSGEN.COM IS YOUR SYSGEN.COM AND NRELOC.COM IS THE REPLACEMENT FOR RELOC.COM (ALSO CALLED CPM.COM AND MOVCPM.COM IN DIFFERENT REFERENCES IN THE DIGITAL RESEARCH MANUALS) DESCRIBED IN SECTION I.

CAREFULLY REVIEW THE SEPARATE BOOKLETS DESCRIBING ED, THE CONTEXT EDITOR, DDT, THE DYNAMIC DEBUGGING TOOL AND ASM, THE ASSEMBLER. ALL OF THE FEATURES AND USER RESPONSES DESCRIBED IN THE MANUALS WILL BE FULLY REPRODUCED IN THIS CP/M ON NORTH STAR DISK IMPLEMENTATION.

BRING IN THE EDITOR AND TYPE IN A SIMPLE FILE. GO THRU THE SEQUENCE OF MAKING A NEW EDIT FILE, ENDING THE EDIT, TYPING THE RESULT WITH THE TYPE COMMAND, AND EXAMINING THE .BAK FILE CREATED. YOU MIGHT TRY MAKING A VERY SIMPLE SUBMIT (BATCH) FILE WHICH SIMPLY GIVES A DIR AND A STAT COMMAND, PERHAPS REPEATED. THIS WILL GIVE YOU AN IDEA OF HOW SUBMIT WORKS. ITS GOING TO TAKE A WEEK OR TWO (OR THREE!) TO GET THE HANG OF ALL THIS. EVEN THOUGH ITS REALLY A VERY EASY SYSTEM TO USE, IT WON'T SEEM AT ALL EASY WHEN

EVERYTHING IS NEW BUT KEEP AT IT.

WHEN YOU HAVE GOTTEN TO THE POINT WHERE YOU CAN EDIT A FILE THAT THE ASSEMBLER WILL ASSEMBLE, LOAD AND RUN THE PROGRAM, USE DDT FOR DE-BUGGING YOUR PROGRAM, CREATE AND RUN SUBMIT PROGRAMS, USE PIP TO TRANSFER FILES, NRELOC AND NSYSGEN TO RELOCATE THE SYSTEM AND TRANSFER IT TO DIFFERENT DISKS AS A MATTER OF COURSE - THEN YOU HAVE LEARNED THE CP/M SYSTEM. NOW USE IT AND HAVE FUN.

THE PURPOSE OF THIS SECTION IS TO PROVIDE YOU WITH GENERAL NOTES ON THE OPERATION OF CP/M ON NORTH STAR DISK.

1. THE FIRST THING YOU SHOULD DO WITH YOUR CP/M ON NORTH STAR DISKETTE AS RECEIVED IS TO WRITE PROTECT IT. DECIDE NOW NEVER TO WRITE ON THIS DISK SO YOU ALWAYS CAN BE SURE IT IS A CORRECT COPY FOR REFERENCE. USE THE NORTH STAR DOS RD AND WR COMMANDS TO COPY THE DISK TO A BLANK DISKETTE IF YOU HAVE ONLY ONE DRIVE, OR USE THE CD (COPY DISK) COMMAND IF YOU HAVE A TWO DRIVE SYSTEM.

2. WHEN USING BLANK DISKETTES THAT HAVE BEEN INITIALIZED WITH THE NORTH STAR DOS, IT IS NECESSARY TO INITIALIZE THE DIRECTORY AREA TO E5H. ELSE, CP/M WILL THINK THE DIRECTORY HAS ENTRIES IN IT. TO DO THIS, FILL AN 8 PAGE (2K) AREA OF RAM WITH E5H, THEN USE THE NORTH STAR DOS COMMAND:

WR 30 RAM 8 CR

3. YOU CAN IMPLEMENT A READ AFTER WRITE CHECK (RWCHK) WHICH WILL OPERATE AS DESCRIBED IN THE NORTH STAR DOS MANUAL. THE RWCHK BYTE IS AT LOCATION 3DFFH+BIAS (3DFFH FOR A 16K SYSTEM) AND IS NORMALLY SET TO ZERO. IF YOU WANT THE READ AFTER WRITE CHECK DONE, SET THAT BYTE TO 1.

4. THIS IS THE ORGANIZATION OF CP/M ON YOUR NORTH STAR DISKETTE - ADD BIAS TO ADDRESSES IF NOT 16K SYSTEM.

	STARTING RAM ADDRESS		
	RUNNING SYSTEM	NSYSGEN	DISK ADDRESS
PAGE 0	0000H	N/A	N/A
TPA	0100H	N/A	N/A
CCP PART 1	2900H + B	0900H	000 THRU 003
BOOT	2000H	0D00H	004 ONLY
CCP PART 2	2D00H + B	0E00H	005 THRU 008
BDOS	3100H + B	1200H	009 THRU 021
BIOS	3E00H + B	1F00H	022 THRU 027
USER	4400H + B	2500H	028 THRU 029
FILE DIRECTORY	N/A	N/A	030 THRU 037
FILES	N/A	N/A	038 THRU 349

PAGE 0 CONTAINS VARIOUS CP/M-RESERVED ADDRESSES. THE TPA IS THE LOCATION AT WHICH MOST CP/M PROGRAMS LOAD AND RUN. THE PROGRAM USUALLY MAY USE RAM UP TO THE START OF BDOS, OVERLAYING THE CCP MODULE IF NECESSARY. NOTE THAT BOOT IS IN THE MIDDLE OF CCP ON THE DISKETTE BECAUSE OF THE NORTH STAR PROM WHICH BOOTS IN SECTOR 4 ON A COLD BOOT ONLY.

5. YOU HAVE 78K AVAILABLE FOR FILES ON AN EMPTY NORTH STAR DISK, COMPARED TO 240K ON AN EMPTY FULL SIZE FLOPPY. THIS IS IN ADDITION TO THE SYSTEM SPACE (TRACKS 0-2) AND DIRECTORY SPACE (TRACK 3, SECTORS 0-7).

6. YOU SHOULD KEEP IN MIND THAT THE MEMORY REQUIREMENTS FOR CP/M ON NORTH STAR DISK ARE 4K GREATER THAN THE NOMINAL SYSTEM SIZE. THE DISTRIBUTION DISKETTE THAT YOU HAVE RECEIVED CONTAINS A 16K SYSTEM WHICH WILL REQUIRE 20K OF MEMORY WHICH STARTS AT ZERO AND RUNS THRU 4FFFH. REMEMBER TO NRELOC SYSTEMS 4K LESS THAN YOUR AVAILABLE MEMORY. THIS ADDITIONAL 4K AREA IS USED FOR THE SPECIAL NORTH STAR DISK DRIVERS AND BUFFER SPACE.

7. IT IS MOST IMPORTANT TO ALWAYS DO A WARM BOOT (CONTROL C) WHENEVER CHANGING DISKS IN A DRIVE. THIS IS NECESSARY TO UPDATE A MAP WHICH IS KEPT IN MEMORY TELLING CP/M WHICH DISK SECTORS ARE IN USE. CP/M VERSION 1.4 WILL NOT PERMIT YOU TO WRITE TO A REPLACED DISK UNTIL THE WARM BOOT IS PERFORMED. ADDITIONALLY THE STAT UTILITY WILL GIVE FALSE RESULTS PRIOR TO A WARM BOOT AFTER A DISKETTE CHANGE. ONE WARM BOOT, WHILE LOGGED IN ON ANY DRIVE DRIVE, IS SUFFICIENT TO UPDATE THE MAPS FOR ALL DRIVES.

8. IN VARIOUS PARTS OF THIS USER'S GUIDE, INSTRUCTIONS ARE GIVEN FOR USING NORTH STAR DOS COMMANDS TO READ OR WRITE DIRECTLY TO DISK WHEN MAKING PATCHES. LOCATION "RAM", AS USED IN THESE INSTRUCTIONS, MEANS ANY CONVENIENT AREA IN MEMORY THAT YOU CHOOSE TO USE FOR THESE OPERATIONS.

9. YOU WILL NOTE THAT THE FIRST ROUTINE IN THE USER AREA IS AN "INIT" ROUTINE. THIS IS FOR YOUR OWN USE, DEPENDING ON THE REQUIREMENTS OF YOUR HARDWARE. THIS INIT ROUTINE IS CALLED EACH WARM BOOT, AND TWICE ON A COLD BOOT. USE THE INIT ROUINE TO BLANK YOUR TVT SCREEN, SET UP USARTS, RESET LINE PRINTER, OR WHATEVER ELSE YOU NEED TO DO. IF YOU WANT INIT CALLED IN A DIFFERENT MANNER (SAY ONLY ON COLD BOOT) YOU MAY DO THIS BY ASSIGNING A "TOGGLE BYTE" IN YOUR USER AREA THAT YOU EXAMINE EACH TIME THE INIT ROUTINE IS CALLED.

10. FOR YOUR INFORMATION, THIS IS THE NORMAL PROCEDURE USED TO BOOT IN THE CP/M SYSTEM USING THE STANDARD NORTH STAR PROM AS SUPPLIED ON THE NORMAL CONTROLLER BOARD. TO START THE PROCEDURE, USE YOUR MONITOR, FRONT PANEL SWITCHES OR NORTH STAR DOS COMMAND " JP E900 " TO GOTO LOCATION E900H.

THEN THE PROM WILL TAKE OVER AND LOAD IN A SINGLE SECTOR FROM DISK ADDRESS 4 (THIS IS THE 5TH SECTOR ON TRACK 0) INTO MEMORY AT LOCATION 2000H. NOTE THAT 2000H IS NOT ONLY WHERE THE PROM BOOTS IN THE LOADER, IT IS ALSO THE FIRST ADDRESS OF YOUR NORTH STAR DOS. WHEN THE BOOT SECTOR IS IN MEMORY, THE PROM TRANSFERS CONTROL TO IT BY JUMPING TO LOCATION 2004H. THE PROGRAM IN MEMORY BETWEEN 2000H AND 20FFH THEN BRINGS CP/M FROM THE FIRST 3 TRACKS ON DISK INTO MEMORY AT THE PROPER LOCATION (2900H TO 45FFH IN A 16K SYSTEM) AND THEN TRANSFERS CONTROL TO CP/M BY JUMPING TO THE BASE OF CCP (2900H IN 16K SYSTEM). THE SYSTEM WILL THEN SIGN-ON.

E - DISK ERROR MESSAGES

THE PURPOSE OF THIS SECTION IS TO EXPLAIN WHAT HAPPENS ON A DISK READ OR WRITE WHEN AN ERROR OCCURS.

THE CP/M MANUAL ALTERATION GUIDE EXPLAINS THAT, WHEN A DISK ACCESS IS MADE, CALLS ARE FIRST MADE TO SELECT THE DISK UNIT, SET TRACK, SET SECTOR AND SET DMA ADDRESS. THEN A CALL IS MADE TO EITHER THE READ OR WRITE ROUTINE. THESE ROUTINES ARE ALL IN THE BIOS JUMP TABLE AT 3E00H+BIAS. IF THE READ OR WRITE IS SUCESSFUL A ZERO WILL BE RETURNED IN THE A REGISTER. IF UNSUCESSFUL, A 1 IS RETURNED IN THE A REGISTER.

ERRORS MAY BE EITHER HARD OR SOFT. THE BIOS WILL AUTOMATICALLY RETRY SOFT ERRORS 10 TIMES BEFORE GIVING UP AND DECIDING THAT THE ERROR IS A HARD ERROR. IF A HARD ERROR IS ENCOUNTERED, TWO ERROR MESSAGES WILL BE PRINTED AT THE CONSOLE AND THEN THE READ OR WRITE ROUTINE WILL RETURN TO CP/M WITH 1 IN THE A REGISTER AS STATED ABOVE.
THE FIRST ERROR MESSAGE PRINTED AT THE CONSOLE COMES FROM BIOS AND IS SIMILIAR IN FORM TO THE MESSAGE PRINTED BY THE NORTH STAR DOS WITH THE ADDITION OF PRINTING OUT THE DRIVE WHERE THE ERROR OCCURRED AND THE TYPE OF ERROR. AN EXAMPLE WOULD BE:

```
' A:125HD-C '
```

WHERE:
- A: INDICATES DRIVE A
- 12 IS THE TRACK NR (DECIMAL)
- 5 IS THE SECTOR (0 THRU 9)
- HD STANDS FOR HARD DISK ERROR
- -C IS THE ERROR TYPE, NO CRC IN THIS EXAMPLE

TYPES OF ERRORS ARE:
- B BODY (SYNCH CHAR) NOT FOUND
- C CRC ERROR, READ OR VERIFY
- V NO VERIFY AFTER A WRITE

CP/M WILL THEN PRINT ON THE CONSOLE THE MESSAGE

```
'PERMANENT ERROR DISK A'
```

NOTE THAT ACCORDING TO THE CP/M MANUAL, YOU CAN CONTINUE AND IGNORE THE ERROR BY TYPING A CARRIGE RETURN AT THIS POINT, CORRECTING THE BAD DATA LATER IF POSSIBLE.

F - PATCHING THE USER AREA

A TWO PAGE (512 BYTE) USER AREA IS PROVIDED FOR YOUR I/O STARTING AT LOCATION BIOS+600H. THIS IS 3E00H+600H= 4400H IN A 16K SYSTEM. THIS USER AREA IS DESIGNED TO HOLD YOUR PERSONALIZED ROUTINES FOR INIT, CONSOLE INPUT AND OUTPUT, LIST, PUNCH AND READER DEVICES. THERE IS A JUMP TABLE WITH 7 JUMPS AT THE BEGINNING OF THE USER AREA WHICH MUST ALWAYS REMAIN IN THE SAME PLACE, AND IN ORDER. ALL INIT, CONSOLE, LIST, PUNCH AND READER ROUTINES IN CP/M JUMP TO THIS JUMP TABLE - THAT IS WHY IT'S LOCATION CANNOT BE CHANGED. THIS JUMP TABLE SHOULD TRANSFER CONTROL TO YOUR ROUTINES WHICH WILL BE IN THE REMAINDER OF THE TWO PAGE USER AREA, OR IN PROM. STUDY THE SAMPLE I/O DRIVERS SUPPLIED, AND LISTED IN SECTION H OF THIS MANUAL, FOR A FURTHER UNDERSTANDING OF THIS.

SECTION H OF THIS MANUAL, "ASSEMBLY LISTING OF SIMPLE TELETYPE DRIVER", CONTAINS THE DRIVER ACTUALLY INSTALLED IN THE DISTRIBUTION COPY OF CP/M ON NORTH STAR DISK. IT MAY BE POSSIBLE FOR YOU TO USE THESE ROUTINES DIRECTLY. IF NOT, YOU MAY BE ABLE TO SIMPLY CHANGE THE PORT NUMBERS, OR POSSIBLY THE STATUS BITS. (NOTE THAT STATUS BITS IN "SAMPLE" ARE INVERTED - ACTIVE LOW - AS PER THE "OLD MITS STANDARD") THIS PATCH SHOULD BE MADE BY FIRST READING THE EXISTING USER AREA FROM DISK INTO RAM WITH THE NORTH STAR COMMAND:

```
RD 28 RAM 2 CR
```

THEN MAKE YOUR CHANGES AND WRITE THE PATCHED USER AREA BACK TO DISK WITH THE WR COMMAND AS EXPLAINED BELOW.

IF YOU CANNOT USE OR MODIFY THE ROUTINES IN THE GIVEN EXAMPLE THEN IT WILL BE NECESSARY FOR YOU TO ASSEMBLE YOUR OWN I/O ROUTINES IN MEMORY. USE WHATEVER FACILITIES YOU HAVE TO ASSEMBLE OR PATCH AT LOCATION "RAM" YOUR I/O ROUTINES. FOR THE BEGINNING, ONLY ATTEMPT ROUTINES FOR INIT, CONSOLE STATUS, AND CONSOLE INPUT AND OUTPUT. LIST AND PUNCH CAN SIMPLY JUMP TO THE SAME ROUTINE AS CONOUT AND THE READER TO THE CONIN ROUTINES AT THIS TIME. THE IDEA IS TO KEEP IT SIMPLE UNTIL YOU ARE ON THE AIR. REMEMBER ALWAYS START THE USER AREA WITH THE SEVEN JUMPS IN THE CORRECT SEQUENCE!!!!

WHEN YOUR USER AREA IS ASSEMBLED OR PATCHED AT LOCATION RAM, LOAD THE NORTH STAR DOS, THEN INSERT THE CP/M DISK AND USE THIS COMMAND TO WRITE YOUR OWN VERSION OF THE USER AREA FROM LOCATION "RAM" TO THE CP/M DISK.

```
WR 28 RAM 2 CR
```

THIS WILL WRITE TO YOUR CP/M DISK YOUR OWN TWO PAGE USER AREA STARTING AT DISK ADDRESS 28. IF YOU CHANGE THE SIZE OF

YOUR SYSTEM IT WILL BE NECESSARY TO RE-ASSEMBLE YOUR USER AREA ROUTINES TO BEGIN AT THE ADDRESS

USER = BIOS+600H+BIAS

WHERE BIAS IS AS EXPLAINED IN THE CP/M ALTERATION GUIDE AND WOULD BE 0 FOR 16K SYSTEM, 400H FOR 17K SYSTEM, 2000H FOR A 24K SYSTEM, ETC.

G - PATCHING FOR A NON-STANDARD PROM

IF YOU HAVE A NON-STANDARD PROM DO NOT ATTEMPT TO START UP CP/M UNTIL YOU HAVE MADE THE PATCHES OUTLINED IN THIS SECTION. BEFORE MAKING ANY PATCHES, WE URGE YOU TO TO MAKE A COPY OF YOUR ORIGINAL CP/M DISK USING THE NORTH STAR DOS COMMAND "CD" FOR COPY DISK, OR USE THE COMMANDS "RD" AND "WR" TO READ SECTIONS OF THE DISK TO MEMORY AND THEN WRITE BACK TO ANOTHER DISK IF YOU ONLY HAVE ONE DRIVE. WRITE PROTET THE ORIGINAL CP/M DISK AND USE IT FOR BACK-UP PURPOSES ONLY.

WE ASSUME THAT YOUR NON-STANDARD PROM IS DIFFERENT IN ONE OR BOTH OF TWO RESPECTS FROM THE STANDARD VERSION USUALLY SUPPLIED ON THE NORTH STAR BOARD. EITHER THE PROM RESIDES AT A DIFFERENT LOCATION THAN E900H OR IT LOADS THE BOOTSTRAP SECTOR (WHICH IS THE FIRST SECTOR OF NORTH STAR DOS AT DISK ADDRESS 4 ON A NORTH STAR DISK) AT A DIFFERENT MEMORY LOCATION THAN 2000H. IN OTHER WORDS, WE ASSUME IT IS BASICALLY THE SAME PROM EXCEPT IT RUNS AT A DIFFERENT LOCATION THAN E900H AND/OR THE NORTH STAR DOS IT BRINGS IN RUNS AT A DIFFERENT LOCATION THAN 2000H.

WE FURTHUR ASSUME YOUR PROM IS SIMILIAR TO THE STANDARD PROM IN THESE RESPECTS:

0. PROM READS 1 SECTOR AT DISK ADDRESS 4 INTO MEMORY.
1. DISK DRIVE NUMBER STORED AT LOCATION BOOT (2000H IN STANDARD SYSTEM)
2. CURRENT TRACK NUMBER STORED AT BOOT+3
3. PROM SETS UP ITS OWN STACK
4. PROM TRANSFERS CONTROL BY JUMPING TO LOCATION BOOT+4 AFTER LOADING.

IF YOU KNOW ONE OF THE ABOVE ITEMS IS DIFFERENT IN YOUR NON-STANDARD PROM - THEN YOU HAVE A PROBLEM WHICH NEEDS SPECIAL ATTENTION. CONTACT LIFEBOAT ASSOCIATES FOR HELP.

FOLLOWING IS THE PROCEDURE FOR PATCHING YOUR BOOT SECTOR TO ALLOW FOR A NON-STANDARD PROM. TO MAKE THESE PATCHES, FIRST LOAD IN THE NORTH STAR DOS, THEN INSERT YOUR CP/M DISKETTE AND READ THE BOOT SECTOR INTO MEMORY LOCATION "RAM" BY USING THE NORTH STAR COMMAND:

```
RD 4 RAM 1 CR
```

THEN MAKE THE PATCHES IN ONE OR BOTH OF THE NEXT TWO PARAGRAPHS, AND THEN WRITE THE PATCHED BOOT SECTOR BACK TO THE CP/M DISK BY USING THE NORTH STAR DOS COMMAND:

WR 4 RAM 1 CR

IF YOUR PROM IS NOT AT E900H, CHANGE TO FOLLOWING 4 LOCATIONS FROM E9 TO THE HIGH ORDER PAGE ADDRESS OF YOUR PROM. FOR EXAMPLE, IF YOUR PROM RUNS AT F200H, THEN CHANGE E9 TO F2 AT THE 4 LOCATIONS. IT IS NOT NECESSARY TO MAKE THIS PATCH IF YOUR PROM RUNS AT E900H.
CHANGE E9H TO F2H AT THE FOLLOWING LOCATIONS:

RAM+11H, +22H, +33H, +44H

IF YOUR PROM LOADS THE BOOT SECTOR AT A LOCATION OTHER THAN 2000H, CHANGE THE FOLLOWING 5 LOCATIONS FROM 20H TO THE HIGH ORDER PAGE ADDRESS OF YOUR BOOT. THIS WOULD NORMALLY BE THE FIRST HIGH ORDER PAGE ADDRESS OF YOUR VERSION OF NORTH STAR DOS. FOR EXAMPLE, IF YOUR DOS RUNS AT LOCATION 9000H, THEN CHANGE 20H TO 90H AT THE 5 LOCATIONS. IT IS NOT NECESSARY TO MAKE THIS CHANGE IF YOUR DOS RUNS AT 2000H.
CHANGE 20H TO 90H AT THE FOLLOWING LOCATIONS:

RAM+14H, +25H, +36H, +47H, +4AH

AFTER YOU HAVE MADE ONE OR BOTH OF THESE CHANGES, WRITE THE PATCHED BOOT SECTOR BACK TO THE CP/M DISK USING THE WR COMMAND AS EXPLAINED ABOVE. THIS SET OF PATCHES MUST BE MADE AGAIN IF YOU GENERATE A NEW SYSTEM USING THE NRELOC UTILITY.

H - ASSEMBLY LISTING OF SIMPLE TELETYPE DRIVER

```
                        ;SAMPLE USER AREA I/O ROUTINE
                        ;
                        ;TO PATCH USER AREA DO THE FOLLOWING:
                        ;       READ USER AREA INTO MEMORY AT ADDRESS "RAM"
                        ;BY LOADING NORTH STAR DOS, INSERTING CP/M DISK
                        ;AND USING NORTH STAR DOS COMMAND:   RD 28 RAM 2 CR
                        ;       EXAMINE AREA "RAM" AND MAKE YOUR PATCHES.
                        ;THEN WRITE PATCHED USER AREA BACK TO CP/M DISK
                        ;BY USING NORTH STAR DOS COMMAND:   WR 28 RAM 2 CR
                        ;
                        ;THIS IS THE USER AREA SUPPLIED ON A STANDARD DISK
                        ;CONTAINS A SIMPLE TELETYPE DRIVER
                        ;
                        ;EQUATES
0010 =          MSIZE   EQU     16      ;CHANGE THIS FOR DIFFERENT SIZE SYTEMS
0000 =          BIAS    EQU     (MSIZE-16)*1024 ;CHANGES BIAS AND ORG FOR SYSTEM
0003 =          IOBYT   EQU     3
4400            USER    ORG     4400H+BIAS      ;NOTE: =BIOS+600H
                        ;
                        ;JUMP TABLE - CP/M JUMPS HERE FOR I/O
                        ;JUMPS MUST REMAIN HERE, IN SAME ORDER
4400 C31844     INIT    JMP     INITR   ;INITIALIZATION
4403 C32044     CONST   JMP     TTYST   ;CONSOLE STATUS CHECK
4406 C32C44     CONIN   JMP     TTYIN   ;CONSOLE INPUT
4409 C33B44     CONOUT  JMP     TTYOUT  ;CONSOLE OUTPUT
440C C33B44     LIST    JMP     TTYOUT  ;LIST OUTPUT
440F C33B44     PUNCH   JMP     TTYOUT  ;PUNCH OUTPUT
4412 C32C44     READER  JMP     TTYIN   ;READER INPUT
                        ;BEGIN USER DRIVER ROUTINES HERE
4415 000000             NOP ! NOP ! NOP
4418 AF320300   INITR   XRA A ! STA IOBYT       ;STORE 0 AT LOCATION 3H
                        ;INSERT YOUR OWN INIT HERE IF NEEDED BY YOUR SYSTEM
                        ;INIT IS CALLED TWICE ON COLD BOOT AND ONCE EACH WARM BOO
                        ;USER CAN KEEP TRACK IF NEEDED WITH "BEEN HERE BEFORE BYT
441C C9000000           RET ! NOP ! NOP ! NOP
4420 DB00       TTYST   IN      TTS     ;PUT YOUR STATUS ROUTINE HERE
4422 E601               ANI     TTYDA
4424 3E00               MVI     A,0
4426 C0                 RNZ             ;A=0 IF NO CHAR WAITS
4427 2F                 CMA
4428 C9                 RET             ;A=0FFH IF CHAR
4429 000000             NOP ! NOP ! NOP
442C DB00       TTYIN   IN      TTS     ;CONSOLE INPUT DRIVER
442E E601               ANI     TTYDA
4430 C22C44             JNZ     TTYIN
4433 DB01               IN      TTI
4435 E67F               ANI     127     ;STRIP PARITY
4437 C9000000           RET ! NOP ! NOP ! NOP
443B DB00       TTYOUT  IN      TTS     ;CONSOLE OUTPUT DRIVER
443D E680               ANI     TTYBE
443F C23B44             JNZ     TTYOUT
4442 79                 MOV     A,C
4443 E67F               ANI     127     ;STRIP PARITY
4445 D301               OUT     TTO
4447 C9                 RET
0000 =          TTS     EQU     0       ;STATUS PORT
0001 =          TTI     EQU     1       ;INPUT PORT
0001 =          TTO     EQU     1       ;OUTPUT PORT
0001 =          TTYDA   EQU     1       ;DATA AVAILABLE
```

Appendix H RELOCATING CP/M USING NRELOC.COM

Note: the file NRELOC.COM has been included with this diskette, which enables you to generate a CP/M system for any memory size, up to 64K bytes. the command

NRELOC cr

(where cr denotes the carriage-return key) loads the NRELOC.COM program and gives it control. This program then examines the current memory configuration, and produces a new CP/M system which is relocated to the top of the memory (actually, the highest contiguous RAM area is used). The newly constructed CP/M system then gets control, and the system starts with the normal sign-on message.

The command

NRELOC * *

constructs a new version of the CP/M system, but leaves it in memory, ready for a sysgen operation. The message

READY FOR "SYSGEN" OR
"SAVE 38 CPMxx.COM"

is printed at the console upon completion, where xx is the memory size in kilobytes. The operator can then type

NSYSGEN to start the system generation

with the response

GET SYSTEM (Y/N)?n user must respond with "n"

and the message

PUT SYSTEM (Y/N)?y user must respond with y

DESTINATION ON B, THEN TYPE RETURN

Place the new diskette on drive B, and type a return when ready (note that if you answer with an "a" rather than a "y" to the prompt above, NSYSGEN will place the CP/M system on drive A instead of drive B). NSYSGEN will then type

FUNCTION COMPLETE, REBOOTING

The user can then go through the reboot process with the old or new diskette.

The operator could also have typed

SAVE 38 CPMxx.COM

at the completion of NRELOC.COM, which would place the CP/M memory image on disk. In this case, the relocated memory image can be "patched" to include custom I/O drivers, as described in the CP/M Alteration Guide.

Note that the memory size can be given explicitly to the NRELOC.COM program when it is started in order to override the internal mechanisms which determine the amount of memory on the system. In this case, the operator must type

```
NRELOC xx
```

or

```
NRELOC xx *
```

where xx is the memory size in decimal kilobytes. The first form produces a CP/M system which operates in xx kilobytes, and starts the newly created system when the relocation is complete. The second form creates the new system, but leaves it in memory for a sysgen or save operation.

For example, the invocation

```
NRELOC 48 *
```

starts NRELOC.COM, and creates a 48K system in memory. Upon completion, the message

```
READY FOR "SYSGEN" OR
"SAVE 38 CPM48.COM"
```

is typed. The operator can then perform the sysgen or save operation as described above. Note that the newly created system is serialized with the number attached to your original diskette, and are subject to the conditions of the Software Licensing Agreement included in this package.

SPECIAL NOTES FOR CP/M ON NORTH STAR DISK OWNERS

1. Remember you need 4K more memory than the nominal size of the relocated system. i.e., A 48K CP/M system requires 52K of memory.

2. "NRELOC 16" will produce "invalid memory size" error message. To generate a 16K system you must give command:

   ```
   NRELOC 16 *
   ```

   Then save the memory image with NSYSGEN. This only applies for generating a 16K system. Any larger system will move itself to execution address and run immediately.

3. If you have changed the user area I/O routines, then the user area will not relocate. You must produce a relocated memory image with NRELOC, then save on disk with NSYSGEN, then assemble your user area - see the users guide - and save on disk using the North Star Dos Command:

   ```
   WR 28 RAM 2 CR
   ```

   where "RAM" denotes the area in memory you stored the memory image of your assemble user area, ready for transfer to disk.

CP/M BASIC Benchmark Tests

Written by W. A. Burton

INTRODUCTION

The purpose of this article is to aid the customers of Lifeboat Associates in selecting an optimal BASIC for their particular application. There is no perfect system for choosing the best possible BASIC, yet a reasonably thorough set of benchmark tests with explanation can greatly ease the difficulty of choice.

The tests were conceived to give every BASIC a fair trial. Both the tests and the author's commentary are offered as impartially as possible. The benchmarks which follow do not purport to predict how a particular BASIC will suit its ultimate environment.

Some general information about the tests:

The BASICS tested were BASIC-E (compiler version 2.0 - run package version K2.0), CBASIC (compiler version 1.01 - run package 1.03), CBASIC (compiler version 2.0 - run package 2.0), XDB (Xitan Disk BASIC version 1.06), MBASIC (Microsoft Disk BASIC version 4.51) and XYBASIC (from Mark Williams Co. - Extended CP/M version 2.4). it is important to note that version 2.0 of CBASIC has been advertised as CBASIC-2. Although the correct name for this product is still CBASIC, to spare confusion it will be called CBASIC-2 in this article. The premise for selection of these particular BASICS is that they are, as of this writing, (12,78) 'State of the Art BASICS' written for CP/M compatability. No doubt better BASICS will emerge, but for now these are the best.

The fourteen benchmarks used were written to run unmodified with all BASICS tested. All tests were run on identical Z80 systems with 64K of memory. All (3) systems operated at 2MHZ with no wait states. All tests were timed by a Canada CL-2400 clock polled by identical software. For all the tests, the return from the subroutine at line 1000 is the beginning of the timing, and the call to the subroutine at line 2000 is the effective end of the timing portion (see programs).

A minor delay is involved with reading four ports for minute and second values. After many trials with a stopwatch, it was determined that this time lag was insignificant. Timings to 1/10 of a second were tried earlier and then abandoned as they seemed to imply statistical importance which these tests do not presume.

BASIC BENCHMARK TESTS AND FEATURES part 1

Any benchmark is nothing more than the 'brainchild' of its author. With little work, such evaluation can vindicate the prejudices of whomever wrote it. Much of the time spent on this project has been devoted to minimizing this common problem.

Notice that all index variables have been zeroed before timing begins. This does in fact improve the performance of some of the interpreter BASICS, yet it is defensible as good programming practice. Other tricks to improve the performance of the interpreter BASICS and CBASIC-2 have been intentionally bypassed. One minor (but necessary) concession has been made to the rule of only including tests which run 'as is' on all six tested BASICS; all but XYBASIC use the INP command to read input ports. In this one case the command IN was substituted. This did not compromise the results, as reading ports is merely a part of the clocking routine which does not noticeably affect the timed portion of any test.

Ideally all testing would have been done on a single computer. This was not possible as no available system was licensed to run all six BASICS. This did not affect any results.

All tests assumed the default precision of each particular BASIC. CBASIC and XDB both default to double precision, and therefore run more slowly than if single precision could have been declared (it can't). MBASIC, XYBASIC and CBASIC-2 allow variables to be declared as integer which would have hastened execution, and MBASIC allows variables to be declared as double precision although not without some very slight speed degradation.

BASIC BENCHMARK TESTS AND FEATURES part 1

Some observations on the six BASICS:

CBASIC, CBASIC-2 and BASIC-E are 'compiler' BASICS. The two versions of CBASIC are sophisticated and specialized products which evolved from BASIC-E. These three offer one major advantage over all the others (except possibly XDB), that is the ability to produce 'secure' unlistable programs. BASIC-E is a good general purpose BASIC which has some considerable limitations. CBASIC and CBASIC-2 are sold as 'commercial' or business BASICS; and this is an honest representation of these products. For commercial business applications, CBASIC and CBASIC-2 are well conceived packages. They excel for such purposes, but are poor by comparison for most others. Where speed of program development and execution are important, these would not likely be the best choices. The syntax and conventions of these BASICS are unique to the point of making it most difficult to readily adapt programs one might find in circulation. This and the non-interactive nature of these three BASICS suggest that they might prove a hindrance to less than experienced programmers. Especially discouraging is the lack of introductory text books adapted to these 'compiler' BASICS. Finally, these are not true compilers. A true compiler produces executable machine code. These BASICS 'pre-compile' source code to a form acceptable to their respective run-time 'interpreters'.

MBASIC and XDB are two excellent general purpose BASICS. Each includes many enhancements over 'minimal BASICS'. Features such as automatic line numbering, renumbering and line oriented editors make these two interactive BASICS the easiest (of the six tested) with which to develop and debug software. If confined to a single 'all-purpose' programming tool, one might be well advised to consider one of these.

XYBASIC is also an interpreter BASIC. There exists a Compile/Run version, but it was not available for these tests. This BASIC in many ways resembles MBASIC and XDB. Many of the commands and options of those BASICS are incorporated here; many are not! The 'convenience' features of MBASIC and XDB such as AUTO, RENUM[ber], EDIT and PRINT USING are not included. Instead of these, many unique features have been incorporated. Included are many commands which allow the most elaborate binary and bit operations of any current microcomputer BASIC. Therein lies the true power of XYBASIC. It is a unique high-level language suited for job or process control. Software for tasks usually done in assembler such as monitoring machinery or interfacing A/D devices can now be developed with the ease of simple BASIC programming. For such applications XYBASIC shines. For business, games or any trivial application, XYBASIC would likely be the least useful of those tested.

NOTE: Just before this article was completed, the author was contacted by a representative of the Mark Williams Co. who promised the imminent release of more enhanced versions of XYBASIC. These forthcoming versions may be very well suited to more generalized development tasks.

One final subjective observation. The only good documentation which accompanies any of these BASICS is the XYBASIC manual. Part of the not unsubstantial price one pays for XYBASIC must be for the excellent and carefully planned user's guide. This must be mentioned as the time saved by good instructional material is usually considerable! Documentation for CBASIC and CBASIC-2 is just adequate, and a good book on the subject of BASIC-E programming is available from the Jem Company in San Francisco. The documentation for the others is substandard by almost any definition.

BASIC BENCHMARK TESTS part 2

```
*****************************
*** THESE ARE THE TESTS: ***
*****************************
```

NOTE : The following lines of code (lines 500-2050) appear identically in all tests except as mentioned for XYBASIC. Although only reproduced once here, these lines must be appended to all the tests to exactly duplicate the benchmarks used.

```
500 GOSUB 2000
501 STOP
1000 B1=INP(169):B2=INP(170):B3=INP(171):B5=INP(173):RETURN
2000 D1=INP(169):D2=INP(170):D3=INP(171):D5=INP(173)
2010 M=10*B5+B1:S=10*B2+B3
2020 PRINT "START --";M;":";S
2030 M=10*D5+D1:S=10*D2+D3
2040 PRINT "DONE  --";M;":";S
2050 RETURN
```

```
100 REM                 TEST01
110 REM                 SIMPLE LOOPING TEST
120 X=0 : GOSUB 1000
130 FOR X=1 TO 10000
140 NEXT X
(+lines 500-2050)       SEE NOTES REGARDING THIS TEST !
```

```
100 REM                 TEST02
110 REM                 SQU. ROOTS, EXPONENTIATION & ROUNDING ERRORS
120 X=0 : Y=0 : Z=0 : CO=0 : GOSUB 1000
130 FOR X=1 TO 300
140 Y=SQR(X)
150 Z=Y^2
160 IF Z<>X THEN CO=CO+1
170 NEXT X
180 PRINT "NUMBER OF ROUNDING ERRORS"; CO
(+lines 500-2050)
```

BASIC BENCHMARK TESTS part 2

```
100 REM                 TEST03
110 REM                 STRING FUNCTIONS
120 X=0 : V=0 : GOSUB 1000
130 A$="0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ" : Z=LEN(A$)
140 FOR V=1 TO 25
150 FOR X=1 TO Z
160 B$="" : C$="" : D$="" : Y=Z-X
170 B$=LEFT$(A$,X)
180 C$=RIGHT$(A$,X)
190 D$=MID$(A$,X,Y)
200 NEXT X
210 NEXT V
(+lines 500-2050)
```

```
100 REM                 TEST04
110 REM                 STRING/NUMERIC CONVERSIONS
120 X=0 : B=0 : GOSUB 1000
130 FOR X=1 TO 1000
140 A$=""
150 A$=STR$(X)
160 B=VAL(A$)
170 NEXT X
(+lines 500-2050)
```

```
100 REM                 TEST05
110 REM                 GOSUB & RETURN
120 X=0 : GOSUB 1000
130 FOR X=1 TO 10000
140 GOSUB 170
150 NEXT X
160 GOTO 500
170 RETURN
(+lines 500-2050)
```

```
100 REM                 TEST06
110 REM                 GOTO STATEMENT (WITHIN AND EXITING FROM LOOP)
120 X=0 : GOSUB 1000
130 FOR X=1 TO 10000
140 GOTO 170
150 NEXT X
160 GOTO 500
170 GOTO 150
(+lines 500-2050)
```

BASIC BENCHMARK TESTS part 2

```
100 REM                 TEST07
110 REM                 SIMPLE ARITHMETIC
120 J=0 : I=0 : A=0 : B=0 : C=0 : D=0 : GOSUB 1000
140 FOR I=1 TO 25
150 FOR J=1 TO 50
160 A=A+I
170 B=B-I
180 C=I*J
190 D=I/J
200 NEXT J
210 NEXT I
(+lines 500-2050)
```

```
100 REM                 TEST08
110 REM                 LOGICAL (BOOLEAN) OPERATIONS
120 X=0 : A=0 : B=0 : C=0 : D=0 : E=0 : F=0 : Z=255 : GOSUB 1000
140 FOR X=0 TO Z
150 A=X AND Z : D=A AND A : D=D AND Z
160 B=X OR Z : E=B OR D : E=E OR Z
170 C=NOT Z : F=NOT C : F=NOT F
180 NEXT X
(+lines 500-2050)
```

```
100 REM                 TEST09
110 REM                 TRANSCENDENTAL, LOG & EXP FUNCTIONS
120 X=0 : A=0 : B=0 : C=0 : D=0 : E=0 : F=0 : GOSUB 1000
130 FOR X=1 TO 87
140 A=SIN(X)
150 B=COS(X)
160 C=TAN(X)
170 D=ATN(X)
180 E=LOG(X)
190 F=EXP(X)
200 NEXT X
(+lines 500-2050)
```

```
100 REM                 TEST10
110 REM                 NUMERIC SWAPS
120 X=0 : A=1 : B=2 : C=0 : GOSUB 1000
130 FOR X=1 TO 3000
140 C=A
150 A=B
160 B=C
170 NEXT X
(+lines 500-2050)
```

BASIC BENCHMARK TESTS part 2

```
100 REM                 TEST11
110 REM                 STRING SWAPS
120 X=0 : A$="1" : B$="2" : C$="" : GOSUB 1000
130 FOR X=1 TO 3000
140 C$=A$
150 A$=B$
160 B$=C$
170 NEXT X
(+lines 500-2050)
```

```
100 REM                 TEST12
110 REM                 CONDITIONAL TESTS
120 X=0 : A=1 : B=2 : C=0 : GOSUB 1000
130 FOR X=1 TO 1000
140 IF A>B THEN C=-1
150 IF A=B THEN C=0
160 IF A<B THEN C=1
170 NEXT X
(+lines 500-2050)
```

```
100 REM                 TEST13
110 REM                 PSEUDOMATRIX OPERATIONS
120 K=0 : J=0 : I=0 : GOSUB 1000
130 DIM A(10,10,10), B(10,10,10), C(10,10,10)
140 FOR I=1 TO 10
150 FOR J=1 TO 10
160 FOR K=1 TO 10
170 A(I,J,K)=-1
180 B(I,J,K)=0
190 C(I,J,K)=1
200 NEXT K
210 NEXT J
220 NEXT I
(+lines 500-2050)
```

BASIC BENCHMARK TESTS part 2

```
100 REM                 TEST14
110 REM                 PSEUDOMATRIX ARITHMETIC & DATA SWAPS
120 K=0 : J=0 : I=0 : GOSUB 1000
130 DIM A(5,5,5), B(5,5,5), C(5,5,5), D(5,5,5), T(5,5,5)
140 FOR I=1 TO 5
150 FOR J=1 TO 5
160 FOR K=1 TO 5
170 A(I,J,K)=A(I,J,K)+(I+J+K)
180 B(I,J,K)=(I*J*K)-A(I,J,K)
190 C(I,J,K)=B(I,J,K)/A(I,J,K)
200 D(I,J,K)=B(I,J,K)*C(I,J,K)
210 T(I,J,K)=A(I,J,K)
220 A(I,J,K)=B(I,J,K)
230 B(I,J,K)=C(I,J,K)
240 C(I,J,K)=D(I,J,K)
250 D(I,J,K)=T(I,J,K)
260 NEXT K
270 NEXT J
280 NEXT I
(+lines 500-2050)
```

BENCHMARK DATA part 3

In the introduction, mention was made of how these tests were created with the intention of neither helping nor handicapping the interpreter BASICS or CBASIC-2. Some very minor changes in the programs for these BASICS can have profound effects on the speed with which they execute. Refer back to TEST01:

```
120 X=0 : GOSUB 1000                (original)
120 GOSUB 1000                      (change 1)
120 DEFINT X : X=0 : GOSUB 1000     (change 2)
120 DEFINT X : GOSUB 1000           (change 3)
130 FOR X%=1 TO 10000               (change 4)
140 NEXT X%                         (change 4)
```

The first line 120 above appears as in TEST01. Had change 1 been substituted, MBASIC would have taken 4 seconds more to run, XDB would have been slowed by 2 seconds and XYBASIC would have have completed 5 seconds faster. Change 2 would have shaved 7 seconds from MBASIC'S original time of 20 seconds, and XYBASIC would have completed the test in 18 rather than 32 seconds. Running the test as per change 3 produced timings of 15 seconds for XYBASIC and 17 seconds for MBASIC. Altering TEST01 as per change 4 reduced the execution time of CBASIC-2 from 86 to 28 seconds. These optimized timings will be included in the results as TEST 1A.

What can be determined from this is that familiarity with the subtleties of these BASICS can make for more (time) efficient software. This becomes even more evident with large programs.

How data from benchmarks is represented:

For each of the fourteen tests the data will be presented as follows: The BASICS will be listed from fastest to slowest for each test. Next will be listed the time (in seconds) each took to finish, then the 'score' which is derived as follows: Whichever BASIC executes a task in the least time is given the arbitrary score of 100. The time this BASIC took to completion is divided in turn by each of the other (slower) times. Each such value multiplied by 100 and rounded to the nearest integer is the respective score for any runner-up in a given test. Finally, when present, a number in braces, e.g. {2}, means look for note {2} at the end of the test in which it appears.

The following abbreviations shall be used throughout:

MBASIC	MBS
CBASIC	CBA
BASIC-E	BSE
XDB	XDB
XYBASIC	XYB
CBASIC-2	CB2

For example the following hypothetical lines:

MBS	26	100
XDB	29	90
etc...		

are interpreted as follows: MBASIC finished the test in 26 seconds, which was the fastest for this test. The score of 100 is assigned to the 1st place finish. XDB was next fastest, and its time of 29 seconds was 90 % as efficient (rounded to nearest whole percent) as the best time. Etc...

TEST RESULTS part 3

******* TEST 01 *******

MBS	20	100
XDB	21	95
XYB	32	63
BSE	55	36
CBA	75	27
CB2	86	23

******* TEST 1A {1} ***

MBS	13	100
XYB	15	87
XDB	21	62
CB2	28	46
BSE	55	24
CBA	75	17

{1} These are the results of TEST01 modified to improve execution speed of the BASICS responsive to such adjustments.

******* TEST 02 *******

XYB	27	100
MBS	30	90
BSE	39	69
XDB	73	37
CBA	277	10
CB2	281	10

******* TEST 2A {2} ***

XYB	221	100
XDB	250	88
MBS	265	83
CB2	275 {3}	80
CBA	275 {3}	80
BSE	282	78

{2} This test counts the number of times from 1 to 300 that the square root of a number raised to the 2nd power does not equal that original number.
{3} Tie

TEST RESULTS part 3

******* TEST 03 *******

XDB	33	100
MBS	37	89
BSE	39	85
XYB	44	75
CB2	46	72
CBA	48	69

******* TEST 04 *******

CBA	21	100
CB2	23	91
MBS	26	81
XDB	29	72
XYB	40	53
BSE	54	39

******* TEST 05 *******

MBS	34	100
XDB	37	92
XYB	57	60
BSE	61	56
CBA	82	41
CB2	91	37

******* TEST 06 *******

MBS	35	100
XDB	40	88
BSE	58	60
XYB	61	57
CBA	79	44
CB2	100	35

******* TEST 07 *******

MBS	29	100
BSE	30	97
XDB	34	85
XYB	47	62
CBA	96	30
CB2	110	26

TEST RESULTS part 3

******* TEST 08 *******

XDB	11	100
MBS	12 {4}	92
BSE	12 {4}	92
XYB	19	58
CBA	33 {5}	33
CB2	33 {5}	33

{4} Tie
{5} Tie

******* TEST 09 *******

MBS	15	100
XYB	18	83
BSE	39 {6}	38
XDB	39 {6}	38
CB2	130	12
CBA	146	10

{6} Tie

******* TEST 10 *******

XDB	22	100
MBS	28	79
CB2	33 {7}	67
BSE	33 {7}	67
CBA	38	58
XYB	44	50

{7} Tie

******* TEST 11 *******

XDB	26	100
MBS	27	96
BSE	35	74
CBA	41	63
XYB	45	58
CB2	46	57

TEST RESULTS part 3

******* TEST 12 *******

XDB	14	100
CBA	16 {8}	88
MBS	16 {8}	88
BSE	16 {8}	88
CB2	24	58
XYB	25	56

{8} Tie

******* TEST 13 *******

XDB	30	100
MBS	32	94
BSE	45	67
CBA	51	59
XYB	54	56
CB2	69	43

******* TEST 14 *******

XDB	24	100
BSE	32	75
MBS	34 {9}	71
CB2	34 {9}	71
CBA	42	57
XYB	44	55

{9} Tie

These are the total times for all fourteen tests:

MBS	375	100
XDB	433	87
BSE	548	68
XYB	557	67
CBA	1045	36
CB2	1106	34

These are the total of the scores for fifteen tests (including 2A). The first number is the total of fifteen scores and the second is the average of those scores.

MBS	1363	91
XDB	1295	86
BSE	1021	68
XYB	986	66
CBA	769	51
CB2	715	48

SPECIAL FEATURES OF TESTED BASICS

The benchmark tests have omitted features not common to all tested BASICS. It is to be emphasized that regardless of how any one BASIC fared in these tests, there is indeed justification for the existence of each. The following section will summarize some of the more important options which could not be tested due to non-conformity with other BASICS. The implementation of special features in certain BASICS may well be of more importance than their apparent execution speed. Such special features will be summarized below preceded by a number with which each can be cross referenced in the table which follows this section. A few of the more obscure features may not have been mentioned here, but it is hoped that the information conveyed will be of help to those debating the merits of different BASICS.

1: 8080/Z80 compatability
2: Array default - DIM N(x) not needed if x less than 11
3: AUTO - automatic line numbering
4: BCD - BCD variables & constants
5: Binary variables & constants
6: CHAIN used with COMMON statetment (see part 5)
7: Compiles to intermediate code (see Secure source codes)
8: COPY - moves text from within BASIC
9: COSH - hyperbolic cosine
10: Data files can be saved on disk
11: DELAY - software programmable & calibratible time delay
12: Descriptive error messages
13: Direct mode - use BASIC as 'calculator'
14: Disable program abort from console
15: Double Precision
16: ELSE clause with IF/THEN
17: EQV - bitwise equivalence
18: Error disabling
19: Error trapping - user defined
20: EXCHANGE (SWAP) - single command to swap variables
21: Extensive (very extensive) binary operations
22: FIX - FIX(X)=SGN(X)*INT(ABS(X))
23: FIX$ - forces strings to specified length
24: FORTRAN like relational operators
25: Global line editor
26: Hex variables & constants
27: IMP - logical bitwise implication
28: INCLUDE (MERGE) - read canned segments into source
29: INSTR - locates 1st position of substring in string
30: Integer division (\) - complement of MOD operator
31: Interrupt processing callable from BASIC
32: Line Editor
33: LOG10 - in addition to 'LOG n' function
34: Logical Unit reassignment
35: Long (and significant) variable names
36: LVAR - lists all variables and their values when called
37: Machine language interface
38: MATCH essentially the same as INSTR

BASIC BENCHMARKS AND FEATURES part 4

39: Matrix Read/Write via single BASIC command
40: MOD - Returns remainder from integer division
41: Multiline user defined functions
42: NULL - puts nulls on paper tape; slows console I/O
43: Octal variables & constants
44: PEEK - examines memory from BASIC
45: POKE - alters selected bytes in memory from BASIC
46: PRECISION - specifies number of decimal digits to output
47: PRINT USING - enables formatted printing
48: Prompt character disable - suppresses '?' after INPUT
49: Provision for interface to hard disk
50: QUOTE - allows choice of string delimiters
51: RENUM(ber) - allows reseguencing of program lines
52: RESTORE w/line number - (re)reads any DATA statements
53: RESUME w/line number - branch for error recovery
54: Secure source codes
55: SINH - hyperbolic sine
56: SIZE - BASIC command returns size of any file on disk
57: STRING$ - command to repeat a string character
58: TIME & DATE can be accessed from BASIC (with clock)
59: TRACE - trace program execution for debugging
60: User definable error codes
61: VARPTR (VARADR) - returns address of variables in memory
62: WAIT - Suspends execution until specified condition is met
63: WHILE/WEND - a controlled structure as in PL/M or Pascal
64: WIDTH - allows console width to be explicitly declared
65: XREF - cross reference utility
66: [?] and ['] - PRINT & REM entry with single keystroke

FEATURES OF TESTED BASICS part 5

The tabulated information concerning the features of the tested BASICS can be interpreted from the following chart as follows:

	MBS	CBA	BSE	XDB	XYB	CB2
44:$	YES	YES	no	YES	YES	YES

Referring back to the chart (part 4) we see that number 44 refers to the PEEK command. Reading across, one can see that only BASIC-E lacks this feature. Some of the features mentioned in the chart deserve more elaboration than a simple YES or no. The '$' which appears after the line number (44:$) indicates that there will be further explanation in the notes following this section.

def	This means 'YES by default'
n/a	This means 'NOT applicable'
opt	This means 'YES as option'

	MBS	CBA	BSE	XDB	XYB	CB2
1:	YES	YES	YES	YES	no	YES
2:	YES	no	no	YES	no	no
3:$	YES	n/a	n/a	YES	no	n/a
4:	no	no	no	no	YES	no
5:	no	no	no	no	YES	YES
6:$	no	no	no	no	no	YES
7:$	no	def	def	no	opt	def
8:	no	no	no	YES	no	no
9:	no	no	YES	no	no	no
10:$	YES	YES	YES	YES	no	YES
11:	no	no	no	no	YES	no
12:	YES	no	no	YES	no	no
13:	YES	no	no	YES	YES	no
14:	no	no	no	YES	no	no
15:$	YES	YES	no	YES	no	YES
16:	YES	YES	YES	YES	no	YES
17:	YES	no	no	YES	no	no
18:	YES	no	no	YES	no	no
19:	YES	no	no	YES	no	no
20:	YES	no	no	YES	no	no
21:$	no	no	no	no	YES	no
22:	YES	no	no	YES	no	no
23:	no	no	no	YES	no	no
24:	no	opt	opt	no	no	opt
25:	no	n/a	n/a	YES	no	n/a
26:	YES	no	no	YES	YES	YES
27:	YES	no	no	YES	no	no
28:	YES	YES	no	YES	no	YES
29:	YES	YES	no	YES	YES	YES
30:	YES	no	no	YES	YES	no
31:	no	no	no	YES	YES	no

FEATURES OF TESTED BASICS part 5

32:$	YES	n/a	n/a	YES	no	n/a
33:	no	no	no	YES	no	no
34:	no	no	no	YES	YES	YES
35:	no	YES	YES	no	YES	YES
36:	no	no	no	YES	no	YES
37:$	YES	YES	no	YES	YES	YES
38:	YES	YES	no	YES	YES	YES
39:	no	no	no	YES	no	no
40:	YES	no	no	YES	YES	no
41:	no	no	no	YES	no	YES
42:	YES	no	no	YES	YES	no
43:	YES	no	no	no	no	no
44:$	YES	YES	no	YES	YES	YES
45:$	YES	YES	no	YES	YES	YES
46:	no	no	no	YES	no	no
47:$	YES	YES	no	YES	no	YES
48:	no	def	no	YES	no	def
49:	no	no	no	YES	no	no
50:	no	no	no	YES	no	no
51:$	YES	n/a	n/a	YES	no	n/a
52:	YES	YES	no	YES	YES	YES
53:	YES	no	no	YES	no	no
54:$	no	def	def	YES	opt	def
55:	no	no	YES	no	no	no
56:	no	YES	no	no	no	YES
57:	YES	no	no	YES	no	no
58:$	no	no	no	YES	no	no
59:$	YES	YES	no	YES	YES	YES
60:	YES	no	no	YES	no	no
61:$	YES	no	no	YES	no	YES
62:	YES	no	no	YES	YES	no
63:	no	YES	no	no	no	YES
64:	YES	YES	no	YES	YES	YES
65:$	no	no	no	no	no	YES
66:	YES	no	no	YES	YES	no

FEATURES OF TESTED BASICS part 5

Notes on the features of the BASICS:

3: (also 32 & 51) These features make the creation of a source file much easier.

6: The CHAIN command of CBASIC-2 allows a program to execute a different program directly. In this regard,it resembles the RUN in MBASIC or the LOADGO command of XDB. However if the COMMON statement is used with the CHAIN command, parameters can be passed between programs without having to use intermediate disk storage. To some this will be invaluable.

7: (also 54) It is possible that someone may be able to defeat the PRIVACY command of XDB.

10: Data can be saved in either random or sequential files.

15: CBASIC-2 describes 14 digit accuracy as 'real'.

21: To list all these options is beyond the scope of this article. To find out all that is available, one should obtain the XYBASIC user's manual.

37: XYBASIC implements this more thoroughly than the others.

44: (also 45) These commands are easier to use if Hex or Octal constants are also available.

58: No mention is made in the manual about which clock(s) might allow these commands to work.

59: XYBASIC has the cleverest implementation of this feature in contrast to the clumsier approaches of both CBASICS.

61: The SADD% function of CBASIC-2 allows the address of string variables in memory to be returned.

65: The XREF facility included with CBASIC-2 produces a listing not unlike the symbol tables of a FORTRAN compiler. Its use as a debugging and maintenance aid is obvious.

APPENDIX author's notes

Prior to this project, I was only familiar with three of the six tested BASICS. While familiarizing myself with the others (and learning more about those I had used), I learned a few things which I will gladly pass along.

BASIC-E and CBASIC do not vary much in speed of execution no matter what is tried.

Most of what BASIC-E can compile can be executed by the CBASIC run-time interpreter as if CBASIC had done the compiling. Exceptions are the few items in BASIC-E which were abandoned when that package was upgraded to CBASIC. This means that a single INT file can execute as either a CBASIC or BASIC-E program depending solely on the run-time package selected.

The absence of a line editor in XYBASIC is a negligible flaw if one has either MBASIC or XDB available for interpretive editing. This also applies to the AUTO and RENUM[ber] commands. One user of XYBASIC does all file preparation with MBASIC and the CP/M text editor. This approach is quite efficient.

A good introductory BASIC text is titled, BASIC AND THE PERSONAL COMPUTER by Thomas A. Dwyer and Margot Critchfield (Addison-Wesley). The thing that makes this book special is that it is written almost entirely in Altair (Microsoft) style BASIC. This would also help the beginner learn either XDB or XYBASIC.

Several versions of BASIC-E are being distributed. Some are quite bug-free some are not. To my knowledge, no version is being supported. By this I mean that in the unlikely event you discover a problem which lies within BASIC-E rather than in the program which made this bug apparent, nobody will be interested in your discovery. Part of the heritage of this language is: 'you pays your money (little or none) and you takes your chances'.

Large programs highlight the imperfections of all tested BASICS. The error detection routines of the 'compiler' BASICS run amok when numerous and varied errors appear in large files. Certain combinations of errors will cause other errors to be reported where none exist. This is partially acknowledged in the CBASIC manual as regards the DL error; the DL error is merely the tip of that iceberg. ASCII loading of large files is horrendously slow with XYBASIC (the manual so specifies) and XDB (it doesn't). EDITing large MBASIC files is a slow process, which becomes even slower as large files grow. The identical comment applies to the RENUMBER command of XDB.

APPENDIX author's notes

The XREF utility of CBASIC-2 will operate on any ASCII files of the BAS filetype. This program recognizes reserved words of CBASIC-2 and presumes them not to be variables. Reserved words from other BASICS will be cross-referenced. Some clever editing of BAS files other than those for the 'compiler' BASICS may enable this feature to be more generally useful than intended.

Microsoft FORTRAN for CP/M

Review by Alan R. Miller, Contributing Editor

INTRODUCTION

Digital computers consist of many binary memory cells. Each of these cells has only two possible states that can be expressed as: TRUE or FALSE; logic 1 or logic 0; ON or OFF; etc. Many different computer languages have been developed to help programmers convert their ideas into this fundamental binary code.

The programmer encodes concepts into a SOURCE PROGRAM and then uses another computer program to convert this source program into a binary OBJECT PROGRAM that the computer can use. FORTRAN, COBOL, PASCAL, and ALGOL are some of the common computer languages that do this translation.

Each type of computer language is especially suitable for a particular task. A line of a FORTRAN source program such as:

```
Z(I) = SQRT(X(I)**2 + Y(I)**2)
```

may be translated into many lines of computer instructions by a compiler or interpreter. The source program is generally machine independent, so that a sorting program written in BASIC will run on a 6800 microcomputer as well as on an 8080.

In contrast to these high-level computer languages, assembly language is a low-level computer language that is more difficult to use, but produces shorter programs that run faster. And, unlike the higher-level languages, each line of an assembly-language source program will generally produce one computer instruction. Besides being more difficult to use, assembly language has another disadvantage. The source program is usable only with a specific type of computer. This means that a sorting routine written in 8080 assembly language will not run on a 6800 computer.

FORTRAN and BASIC languages are especially suitable for mathematical calculations (compared to COBOL, e.g., which is useful for the handling of business records). These high-level programs utilize a separate processing program to convert the original, user-written source program into the ultimate binary code needed by the computer.

BASIC source programs are commonly processed by a BASIC interpreter that resides in the computer memory along with the user's original source program. Each line of the source program is interpreted as it is encountered. Thus if the instruction:

```
Y(I) = X(I)
```

occurs in a loop that is executed 500 times, the same instruction is interpreted 500 times. Exceptions to this are BASIC-E and CBASIC. For these programs, a preprocessor first converts the source program into an intermediate program, which is then used by a run-time monitor.

ADVANTAGES OF FORTRAN

FORTRAN works a little differently. Each source program is first compiled into a relocatable binary object program. Then a linking loader program places the needed relocatable modules into memory in such a way that they can be run by themselves. No run-time monitor or interpreter need be present. The advantages of FORTRAN compared to BASIC are that less memory is required at run-time and the programs run faster (once they have been compiled) since only the ultimate binary code resides in memory. BASIC requires an 8K to 20K-byte run-time interpreter, as well as the original source code, with all of its comments, to be present in memory. FORTRAN is faster since the source program instructions don't have to be converted each time they are encountered.

A third advantage of FORTRAN, the localization of variables, may be the most important of all. If a subroutine is written to sort an array X of length N, it can readily be used to sort the array Z of length M.

```
DIMENSION X(30),Z(50)
  •  •
CALL SORT (Z,M)
  •  •
CALL SORT (Z,M)
  •  •
SUBROUTINE SORT(X,N)
CIMENSION X(1)
  •
RETURN
```

By contrast, all variables are global in BASIC. This means that the array Z would have to be copied into the array X and N would have to be changed to M before the sort routine could be called a second time:

```
10 DIM X(30),Z(50)
20 M = 50 : N = 30
60 GOSUB 1000 : REM SORT X
100 N = M
110 FOR I=1 TO N
120 X(I) = Z(I)
130 NEXT I
140 GOSUB 1000 : REM SORT Z
 •  •
1000 REM SORTING ROUTINE
```

And if the array X were needed later, it would have to be saved by the first copying into another array. Of course, there could be two sort routines, one for X and the other for Z, but this solution seems to be even worse.

Yet another advantage of FORTRAN is that there is a wealth of software available in the mathematics and engineering fields. For example, the IBM Scientific Subroutine Package contains routines for statistical analysis, curve fitting, and simultaneous solution of linear equations.

One of the greatest disadvantages of FORTRAN is that a program cannot be debugged as easily as a BASIC program. Typing a Control-C will stop a BASIC program while it is running. The user can then print the current values of any of the variables and even change the values. The program can then be resumed with a CONT command. This potential problem can be greatly reduced in FORTRAN, however, by programming in modular fashion. Thus an input subroutine, and output subroutine, a sort subroutine, etc., can each be written, compiled, run, and debugged if necessary. These modules can then be called by a main program when needed.

Another possible problem with FORTRAN is that no check is made to see if array indexes are out of range. Consider the following example:

```
10 = ZLOC(Y), X(10),Y(10)
*  *  *
Y(1) = 5
X(11) = 8
WRITE (1,103) Y(1)
```

The value of Y(1) has been changed from 5 to 8. Y(1) was initially set to 5, but the expression X(11) actually evaluates to 11 locations past the start of X. In this case it is also the address of Y(1). This potential problem is present in almost all versions of FORTRAN.

MICROSOFT FORTRAN

Microsoft, the organization that produced the MITS BASICs, and the TRS Level II BASIC, now offers a disk-based FORTRAN for the 8080 and Z-80 microprocessors. Versions are available for CP/M, Tektronix, ISIS-II, DTC Microfile, and MITS disk operating systems. A net memory size of 24K bytes, in addition to the disk operating system (DOS), is needed for the compiler. The CP/M version is reviewed in this article, but the other versions appear to be similar. The Microsoft CP/M version of FORTRAN is easily implemented since it uses the CP/M DOS primitives for all peripheral operations such as disk, console, list output, etc.

THE MANUALS

Three extensive and well-written manuals are provided with FORTRAN-80:

1. FORTRAN Reference Manual
   Language, grammar, and syntax
2. FORTRAN User Manual
   a. Use of compiler
   b. Run-time error messages
3. Utility Software Manual
   a. Assembler
   b. Linking loader
   c. Library manager
   d. Differences for versions

The total documentation runs for 152 pages and comes in an attractive and useful ring binder.

CREATING A FORTRAN SOURCE PROGRAM

FORTRAN source programs are generated and edited with the regular CP/M context editor:

B>A:ED SORT.FOR

The default extension is FOR. ANSI Standard FORTRAN X3.1966 is utilized except that there are no complex functions. There are also some additional nice features that are discussed later in this article.

The standard FORTRAN line of 80 characters has the format:

Column 1 - 5	Statement label, a decimal number
Column 7 72-	Statement field
Column 72-80	Identification field

And if the statement is too long:

Column 6	Continuation field (next line)
Column 7 -72	Continuation of the statement

Comments can be placed between statements:

Column 1	The letter C
Column 2 -72	Text of the comment

USE OF THE ASCII TAB CHARACTER

The ASCII tab (Control-I) can be used to speed up the typing and reduce the size of the source program. Enter the label (line number) first (if any) starting in column 1. Then type a tab followed by the FORTRAN statement. The compiler will interpret the tab as the equivalent number of spaces. Thus:

12 <tab> X = 4

has the same meaning as:

12 <6 blanks> X = 4

If you have existing FORTRAN source programs that use blanks instead of tabs, they can be converted by using the substitute command in the CP/M editor:

BMS^L ^Z^L^I^Z

(The up-arrow means that the control key is pressed.)

THE ORDERING OF SOURCE STATEMENTS

For subprograms, the first line is a SUBROUTINE, FUNCTION, or BLOCK DATA statement. The next group of statements (and the first group for a main program) are the specification statements. They must appear before any executable statements, and must be in the following order:

```
   EXTERNAL, DIMENSION, REAL, INTEGER, ETC.
COMMON
EQUIVALENCE
DATA
```

The executable statements appear next:

```
A = SQRT(X*X + Y*Y)
IF (I .LT. K) GOTO 28
STOP
```

It is good programming practice to group the format statements after the last executable statement (this will usually be a STOP or RETURN).

```
100  FORMAT(' PARABOLIC FIT')
101  FORMAT(1P6E13.2)
```

The final statement in each program is:

```
END
```

More than one program may be placed into the same file. This would normally be done if there are subroutines used only by one main program, or if one of the subroutines called the others. On the other hand, generaly subprograms such as a sort routine might be called by several different main programs. These then should either be placed into separate files, or combined with several similar routines into a utility library.

ADDITIONAL FORTRAN FEATURES

FORTRAN-80 adds some nice features to the standard ANSI FORTRAN:

1. Logical variables
2. Logical DO-loop indices
3. Mixed-mode arithmetic
4. ASCII strings in expressions
5. Hexadecimal constants
6. Logical operations
7. END= and ERR= in READ and WRITE
8. ENCODE and DECODE
9. PEEK and POKE
10. INP and OUT

FORTRAN considers variables starting with the letters I through N to be integers, and the others to be real, single-precision variables. But this default mode can be over-ridden with specific declarations. Variables can be explicitly declared as one of four types:

LOGICAL	1 byte, with a value of TRUE or FALSE or a number from -128 to 127
INTEGER	2 bytes, -32,768 to 32,767
REAL	4 bytes, 7+ decimal digits 10**-38 to 10**38

DOUBLE PRECISION 8 bytes, 16+ decimal digits; same dynamic range as REAL.

There is effectively a fifth type of variable. Any of the above four variables can be used as a STRING variable, with a maximum of one ASCII character per byte.

MIXED MODE

Mixed-mode arithmetic means that an expression such as:

```
Y = 2 * A + 3
```

is allowed, i.e., the decimal points are not needed on the 2 and the 3. Hexadecimal constants can be defined with either an X or a Z:

```
I = Z'FF'    and
J = X'C0'
```

ASCII strings can be defined in three ways: in a data statement, a replacement statement, or a FORMAT statement.

```
      INTEGER TITLE(10)
      DATA TITLE/'NON-','LINE','AR C', 'URVE',' FIT'/
or
      NO = 'NO'
or
      WRITE (1,101)
            • • •
101   FORMAT('PRESSURE VS. TEMPERATURE')
```

The END= option makes it easy to read data without knowing how much there is. The statements:

```
      READ(6,102,END=20)(A(I),I=1,99)
20    N = I - 1
```

can be used to read values into the array A from logical device 6 (a disk for example) until the end-of-file (EOF) mark is encountered. Then the statement, labeled 20, sets the correct number of items read. (Since the EOF mark was also counted, the total must be reduced by one.)

ENCODE and DECODE operations allow the interconversion of ASCII and numeric values, much like the VAL and ASC functions of BASIC. PEEK and POKE allow memory locations to be read or changed. INP and OUT can be used to communicate with peripherals.

COMPILER THE FORTRAN SOURCE PROGRAM

At this point, the FORTRAN source programs have been generated with the CP/M context editor, or copied to a disk file from paper tape using the CP/M PIP program.

We also use a third method. IBM cards are read into our campus central computer and saved on disk files there. A telephone link is then established to our microcomputer using a modem. The FORTRAN files are transferred over the telephone line into our computer memory starting at 100 HEX. The programs are then saved on a floppy diskette by using the CP/M SAVE command.

It is possible, of course, to proceed this far without actually having a FORTRAN compiler, since only the CP/M editor has been used. You might want to do this in anticipation of receiving FORTRAN if you have a large library of programs.

THE ACTUAL COMPILING

Source programs are compiled with the command:

```
F80 =SORT
```

or

```
A:F80 =B:SORT
```

if the compiler is on drive A and the source program is on drive B. Several programs can be more easily compiled with the command:

```
F80
*=SORT
*=CPLOT
*=B:CONVFIT
*^C
```

In this case, the compiler prompts each new line with an asterisk. A Control-C is used to indicate the end of the compile session. If there are several subprograms within a single file, the compiler will list the name of each subprogram as it encounters it. The filename need not match any of the subprogram names. If the file contains a main program, the word $MAIN will also appear in the list during the compile procedure.

The compiler produces a relocatable, machine-language program with the same name as the source file, but with the file type of REL. During compilation, two types of error messages may be printed: warning and fatal errors. A warning might occur if a STOP statement were temporarily inserted into the middle of a program during a debugging session:

```
      WRITE (1,101) X
      STOP
      X = 4
```

The compiler will discover that there is no way to reach the statement X = 4 and so issues a warning message. Although this is not a serious problem, the warning message can be avoided by adding the dummy statement:

```
100   CONTINUE
```

after the STOP statement.

A fatal error can occur, for example, if there is an odd number of parentheses in a statement:

```
      Y = A * (B + LOG(C)
```

In this case, it will be necessary to correct the error using the CP/M text editor, then recompile the program with F80.

A FORTRAN LISTING FILE

The FORTRAN compiler can be directed to generate a listing file during the compile process. The switch /L is used for this purpose.

```
F80 = SORT/L
```

This causes an additional file, with the extension PRN to be produced. It contains the original lines of the source program with the corresponding assembly listing of the generated code, interspersed throughout.

The PRN file is useful in debugging a program. It can also be used to increase the efficiency of a frequently used subprogram. In this case, the program is first written in FORTRAN, then compiled with the /L switch. Finally, the PRN file can be used as a guide for writing a more efficient assembly language program.

EXECUTING A FORTRAN PROGRAM

When all of the modules have been successfully compiled, they can be executed with the linking loader:

```
L80  MAIN,SUBR1,SUBR2/G
```

where MAIN, SUBR1, and SUBR2 are the file names of relocatable files. (Each may contain several subroutines.) The standard FORTRAN library routines such as ABS, ATAN, EXP, SIN, etc., are located in a file named FORLIB.REL. If FORLIB resides on the currently logged-in disk, it will be automatically searched for the necessary programs. If, however, the user-written FORTRAN programs are on a different drive from the FORTRAN processing programs, then the process is a little more complicated. The drive names must be included and FORLIB must be specifically listed if it is not on the default drive. For example, the execution command can be:

```
A>L80 B:MAIN,B:MATHLIB/G
```

A>v

```
B>A:L80 MAIN,MATHLIB,A:FORLIB/S/G
```

if B is the default drive. Notice that the filetype REL is not entered.

The FORTRAN linking loader will automatically find all necessary programs, relocate them in memory, then start execution if the /G switch has been given. The /S switch immediately following FORLIB instructs the loader to search that library for the necessary routines and then load them into memory. If the /S switch is not given, the entire FORLIB library will be loaded into memory.

The absolute memory image can be saved as a disk file of type COM if the /N switch is set. This will allow the program to be more quickly run. But the disadvantage is that the COM file requires relatively large amounts of disk space.

OUTPUTTING THE DATA

At some point in the process, the programmer will want to see at least some of the results of the calculations. This is accomplished in FORTRAN with a WRITE statement.

```
WRITE (LUN,101)  <list>
```

where LUN is the FORTRAN logical unit number specifying the particular peripheral, 101 is the line number of the format statement and <list> is a list of the variables to be written.

Logical unit numbers 1,3,4, and 5 are preassigned to the system console. An LUN of 2 is preassigned to the list device and LUN values of 5 through 10 are preassigned to disk operations. Units 11 through 255 can also be used by the programmer.

During the development of a new program, it would be advantageous to first view the results on the video screen of the system console. This is accomplished by defining the LUN in the WRITE statements to be 1. Then after the program is running satisfactorily, the output can be sent to the line printer so a permanent copy can be obtained. There are several ways in which this can be accomplished.

If the CP/M IOBYTE feature has been implemented, then the program called STAT can be used to reassign the console output to the list device:

```
A>STAT CON:=LST:
```

When the FORTRAN program is executed again, the results will appear at the line printer.

Another method would be to input the LUN from the console near the beginning of the program.

```
      LUN = 1
      WRITE (1,101)
      READ (1,102) NOYES
      IF (NOYES .EQ. 'Y' .OR. NOYES .EQ. 'y') LUN = 2
            • • •
101   FORMAT(' OUTPUT TO LINE PRINTER? ')
102   FORMAT(A1)
```

This routine only looks at the first character that was entered, ignoring the rest. Thus, inputting a YES, a Y or a YUP will send the output to the line printer. Any other answer will send the output to the console.

ABORTING A FORTRAN PROGRAM

Suppose that you would like to generate a stream of random numbers so that the calculated values can be examined. Then at some point, you would like to stop. A Control-C can be used to abort a BASIC program in this case, but FORTRAN has no such option built in. The INP function provided by Microsoft, however, can be used for this purpose. The following routine could be executed after every 100 loops. It is written for a console status port of decimal 16, and a read-ready flat at bit 0, active high.

```
      DONE = INP(16)
      DONE = DONE .AND. 1
      IF (DONE) STOP
```

DISK INPUT-OUTPUT

Both sequential and random-disk file access are available in the CP/M version. FORTRAN logical unit numbers 6 to 10 have been preassigned for this purpose. The FORTRAN statement:

```
WRITE (6,101) (A(I),I=1,N)
```

will place the data into a file named FORT06.DAT of the currently logged-in disk.

Alternatively, a more specific method is available. The command:

```
CALL OPEN (6,'NEWDATA ASC',2)
```

will open a file named NEWDATA.ASC on drive B and associate it with logical unit number 6. The first argument defines the logical unit number and must evaluate to an integer. The second argument is the filename. Notice that it is not in the usual CP/M format. In this case, the filename must evaluate a string of exactly 11 ASCII characters and must not contain the usual decimal point between the primary name and the extention. The first eight characters are the primary name and the last three characters are the file type. If the primary filename is shorter than eight characters, as in the above example, the remainder must be filled with blanks.

The third argument of OPEN specifies the disk drive, and must evaluate to an integer. A zero value refers to the default drive and the numbers 1 through 4 explicitly specify drives A through D. Once a file has been opened, it can be read with the command:

```
READ (6,102) (B(I),I=1,N)
```

If data is written to the file with the statement:

```
WRITE (6,105) A,B,C
```

then a new file is created. If a file of the same name already exists, it is erased before the new data is written.

At the end of the disk access, the file should be closed with the command:

```
ENDFILE 6     or
REWIND 6
```

The latter command closes the file, then reopens it. This could be used to write data in one format, then read it back in a different format. (But see the ENCODE and DECODE commands.).

ASSEMBLY-LANGUAGE PROGRAMS FOR FORTRAN

The Microsoft FORTRAN compiler converts the user's source program into a relocatable machine-language program which is in turn converted into binary code. But the resulting binary code may not be as fast or occupy as small a memory space as if it had been originally written in assembly language. The tradeoff is that the FORTRAN source program can generally be written and debugged much more rapidly than if it had been written in assembly language. Nevertheless, for short, frequently used subroutines, it is often advantageous to use assembly language rather than FORTRAN.

The Microsoft FORTRAN package contains a macro assembler that produces compatible, relocatable modules that can be called from FORTRAN programs in the usual way. In fact, the programmer will not generally be concerned with whether the relocatable modules were originally written in FORTRAN or in assembly language.

An assembly language format to generate real random numbers can be written, since such a routine is not provided in the standard library. The algorithm, which appeared in the October 12, 1978, issue of Electronics is used to generate a 24-bit integer (Listing 1).

The low-order 23 bits are copied into the 3-byte mantissa of the FORTRAN floating-point accumulator (at $AC). The 24th (high order) bit is zeroed to make the resulting number positive. The 8-bit exponent is set to 80 HEX to give a resulting range of 0.5 to 1.0. The number is then converted to the usual range of 0 to 1 by using the FORTRAN arithmetic routines. The random number is first multiplied by 2 with the subroutine $MA, then 1 is subtracted with the subroutine $SA.

Notice that the subroutines $MA and $SA are declared as external, as is the location of the floating-point accumulator $AC. Also the subroutine name RND, used by the calling FORTRAN program, is declared to be an entry.

The assembly language random-number generator can be called from a FORTRAN program in the usual way:

```
Y = RND(NSKIP)
```

A real random number between 0 and 1 will be placed into the variable Y. The integer argument instructs the function to skip over NSKIP random numbers before choosing the next number. This argument can be retrieved with a MOV A,M instruction, since the H,L register pair points to the least-significant byte of argument.

THE LIBRARY MANAGER

The CP/M version of FORTRAN contains a program called LIB that can be used to build library files of relocatable programs. For example, the relocatable module of the above random-number generator can be incorporated into FORLIB by use of the program LIB. This makes it unnecessary to specifically list the module RND in the link command at execution time.

A SPEED COMPARISON

The Microsoft CP/M version of BASIC is much faster than earlier versions such as 4.1 EXTENDED, and also faster than many of the other 8080 or Z-80 BASICs. A speed comparison was made between Microsoft BASIC and FORTRAN by solving sets of linear equations. The same algorithm, a Householder technique, was coded in both BASIC and FORTRAN. the BASIC statement:

```
DEFINT I-N
```

was used to declare loop variables to be two-byte integers for faster operation. The FORTRAN program consistently produced the solution 8 times faster than the BASIC version (17 seconds vs. 135 seconds for 14 equations).□

PROGRAM LISTING

```
                        ; RND: A FORTRAN-CALLABLE FUNCTION
                        ;      TO GENERATE A RANDOM
                        ;      NUMBER FROM 0 TO 1
                        ;
                        ; METHOD: ELECTRONICS, OCT. 12, 1978
                        ; USAGE:  X = RND(NPASS)
                        ;  NPASS IS THE NUMBER OF TIMES TO SKIP
                        ;
                        ; PROGRAMMED BY ALAN R. MILLER
                        ; NEW MEXICO TECH, SOCORRO 87801
                        ;
0000'                           TITLE   RANDOM-NUMBER GENERATOR
                        ;
0000'                           EXT     $AC,$MA,$SA
0000'                           ENTRY   RND
                        ;
0000' 06 01             RND:    MVI     B,1         ;SET FOR ONE PASS
0002' 7E                        MOV     A,M         ;GET ARGUMENT
0003' E6 0F                     ANI     0FH         ;TAKE 4 BITS
0005' CA 0009 '                 JZ      NEXTN       ;CHANGE 0 TO 1
0008' 47                        MOV     B,A
0009' 2A 0049 '         NEXTN:  LHLD    WORD2       ;HIGH 2 BYTES
000C' EB                        XCHG                ;PUT IN D,E
000D' 2A 0047 '                 LHLD    WORD1       ;LOW 2 BYTES
0010' 29                        DAD     H           ;SHIFT LEFT
0011' 7B                        MOV     A,E
0012' 17                        RAL                 ;SHIFT LEFT
0013' 5F                        MOV     E,A
0014' AD                        XRA     L           ;FEEDBACK
0015' F2 0019 '                 JP      SKIP
0018' 23                        INX     H
0019' 22 0047 '         SKIP:   SHLD    WORD1       ;HIGH 1 BYTES
001C' EB                        XCHG
001D' 22 0049 '                 SHLD    WORD2       ;HIGH 2 BYTES
0020' 05                        DCR     B           ;COUNT
0021' C2 0009 '                 JNZ     NEXTN       ;DO IT AGAIN
0024' 21 0000 *                 LXI     H,$AC       ;POINT TO FAC
0027' 3A 0047 '                 LDA     B1          ;LOW BYTE
002A' 77                        MOV     M,A         ;PUT IN FAC
002B' 23                        INX     H
002C' 3A 004B '                 LDA     B2          ;SECOND BYTE
002F' 77                        MOV     M,A
0030' 23                        INX     H
0031' 3A 0049 '                 LDA     B3
0034' E6 7F                     ANI     7FH         ;BIT 7 PLUS
0036' 77                        MOV     M,A         ;PUT INTO FAC
0037' 23                        INX     H
0038' 36 80                     MVI     M,80H       ;SET EXPONENT
003A' 21 0002                   LXI     H,2
003D' CD 0000 *                 CALL    $MA         ;TIMES 2
0040' 21 0001                   LXI     H,1
0043' CD 0000 *                 CALL    $SA         ;SUBTR 1
0046' C9                        RET
                        ;
0047'                   WORD1:
0047' 0D                B1:     DB      0DH
0048' B1                B2:     DB      0B1H
0049'                   WORD2:
0049' 9B                B3:     DB      9BH
004A' 80                B4:     DB      80H
004B'                           END

$AC     0025*   $MA     003E*   $SA     0044*   RND     0000'
NEXTN   0009'   WORD2   0049'   WORD1   0047'   SKIP    0019'
B1      0047'   B2      004B'   B3      0049'   B4      004A'
```

Microsoft BASIC

Overview

Microsoft BASIC is an extensive implementation of BASIC for 8080 and Z-80 microprocessors. Its features are comparable to those of BASICs found on minicomputers and large mainframes.

Current Versions of Microsoft BASIC

Microsoft BASIC is currently in its fourth major release (4.6). Each release consists of four different versions of BASIC:

1. 4K version: Stripped down version to run in minimum memory. Includes direct statement execution, dynamic dimensioning of arrays and multiple statements per line.

2. 8K version: Standard version. Includes string manipulation and multiple dimension arrays. (also available for 6800 and 650x series MPUs.)

3. Extended version: Requires 15K of memory. Features include integers, double precision, EDIT, AUTO, RENUM, PRINT USING, etc.

4. Disk version: Requires 17K of memory. All features of Extended version plus random and sequential file access on floppy disk.

The different versions are generated from the same source files using conditional assembly switches. Each version is upward compatible with larger versions.

Features

Microsoft BASIC, widely know as Altair BASIC, is the most extensive 8080/Z-80 BASIC available. It contains many unique features rarely found in other BASICs:

1. Direct access to CPU I/O ports (INP, OUT)

2. Ability to read or write any memory location (PEEK, POKE)

3. Matrices with up to 255 dimensions

4. Dynamic allocation and deallocation of matrices at execution time (DIM A[I,J],ERASE A)

5. IF...THEN...ELSE and nested IF...THEN...ELSE

6. Direct (immediate) execution of statements

7. Error trapping

8. Four variable types: Integer, String, Single Precision Floating Point (7-digits) and Double Precision Floating Point (16-digits)

9. Full PRINT USING for formatted output (includes asterisk fill, floating $ sign, scientific notation, trailing sign, comma insertion)

10. Extensive program editing facilities via EDIT line command, RENUM, AUTO, etc.

11. Trace facilities (TRON, TROFF)

12. Ability to call up to 10 assembly language subroutines

13. Boolean operators OR, AND, NOT, XOR, EQV, IMP

14. BASIC can be placed on ROM

Microsoft Disk BASIC also supports files on multiple floppy disks:

1. Sequential files with variable length records

2. Random files (record I/O)

3. Complete set of file manipulation statements: OPEN, CLOSE, GET, PUT, KILL, NAME, etc.

4. Up to 255 files per floppy disk

5. Runs standalone or under CPM, ISIS-II or TEKDOS operating systems

Comparison of BASICs

In our comparisons with other BASICs, we will examine only the Disk version. The features of the other versions may be obtained from the Microsoft BASIC Reference Manual.

An examination of Table I shows that Microsoft BASIC is comparable to RSTS BASIC and generally superior to PDP-10 BASIC and 5100 BASIC in terms of statements, functions and editing facilities. Microsoft BASIC is the faster microprocessor BASIC of the two examined and is close to RSTS BASIC in execution speed (between two and five times slower). This is impressive considering the CPU is a two microsecond LSI chip. (Faster versions of the 8080 CPU are available.) For an in-depth comparison of microprocessor BASIC execution speeds, see "BASIC Timing Comparisons," Kilobaud, October, 1977. This comparison features Microsoft's 8K 6502 BASIC (OSI BASIC and PET BASIC) as well as Microsoft's 8K 6800 BASIC (Altair 680 BASIC) and 8080 8K Extended and Disk BASICs (Altair BASIC).

Microsoft BASIC also uses less memory than any other BASIC examined. For fairly large system programs, microprocessors are proving to be as good or better than minis or large mainframes in terms of efficiency of memory use.

	Microsoft BASIC	DEC RSTS BASIC	DEC PDP-10 BASIC	IBM 5100 BASIC
CPU	8080 or Z-80	PDP-11/50, other 11	KA (or KI or KL) -10	370 compatible
Size (bytes)	17K	28K	67K	48K
Statements	41	40	30	32
Variable types	Integer, String, Real, Double Real	Integer, String, Real	String, Real	String, Real
Numeric Functions	21	13	21	25
String Functions	13	13	9	1
Direct Statements	Yes	Most	No	No
Mass Storage	Floppy Disk, Cassette	Large Disks	Large Disks	3M Tape Cartridge
Implementation	Interpreter	Interpreter+PseudoCode	Compiler	Interpreter
Maximum Program Size	44K	32K	.5 Megabytes	32K
Floating Point Accuracy (Decimal Digits)	7.1 and 16.8	7.1 or 16.8	8.1	13
Time to Execute 10000 Iteration FOR Loop	15 seconds	4 seconds	.03 second	16.4 seconds
Time for 10000 Iteration Integer FOR Loop	7 seconds	3.6 seconds	N/A	N/A
Multi Line Functions	No	Yes	Yes	Yes
Boolean Operators	Yes	Yes	No	No
IF...THEN...ELSE	Yes	Yes	No	No
Multi Statement Lines	Yes	Yes	No	No
PRINT USING	Yes	Yes	Yes	Yes
EDIT line	Yes	No	No	No
RENUMBER	Yes	No	Yes	Yes
Automatic Line Insert	Yes	No	No	Yes
Cost of minimum configuration (approx)	$2000	over $30,000	over $200,000	$8000
Unique Features	Long Variable Names, IMP, XOR, EQV, MOD, Substring Assignment, Hex, Octal constants, PEEK, POKE, INP, OUT	Statement Modifiers, IMP, XOR, EQV, MOD, Virtual Matrices	CHANGE Multiple LET	Special Interrupt Keys

TABLE I

Pricing

Single Copies

All versions of Microsoft's 8080/Z-80/8085 BASIC are available off the shelf. Each user must sign a non-disclosure agreement before the copy of BASIC will be shipped by Microsoft. Updates for enhanced versions will cost between $25 and $75, depending on the extent of the enhancements. Backup copies of BASIC may be purchased for $25. A BASIC manual will be included with every BASIC shipped except for backup copies.

In the memory requirements given below, only the size of BASIC itself is given.

Version	System	Price	Supplied on
8K	Intellec	$150	hex paper tape
8K	MDS	$150	hex paper tape
8K	SBC 80/10	$150	hex paper tape
8K	SBC 80/20	$150	hex paper tape
Extended (15K)	Intellec	$250	hex paper tape
Extended (15K)	MDS	$250	hex paper tape
Extended (15K)	SBC 80/10	$250	hex paper tape
Extended (15K)	SBC 80/20	$250	hex paper tape

Hex paper tapes use standard Intel format.

NOTE: ROM versions for the above may be ordered at $1000 per copy.

Disk (17K)	CP/M	$350	full size single density diskette
Disk (17K)	TEKDOS	$350	full size single density diskette
Disk (17K)	ISIS-II	$350	full size single density or doubl density diskette (please specify)

(Above prices include BASIC Reference Manual.)

BASIC Reference Manual, purchased separately	$ 20

NOTE: Microsoft's 8K 6502 BASIC may be obtained from:

Johnson Computer
P.O. Box 523
Medina, Ohio 44256
216-725-4560

Dealer Purchases

Dealers may purchase CP/M BASIC from Microsoft for $250 per copy if they purchase at least four copies and sign a standard dealer agreement.

OEM Licensing

Both flat fee and royalty licenses may be obtained from Microsoft for any of the above BASICs or for custom versions. For more information on OEM licenses, please contact: Paul G. Allen, Vice President
Microsoft
10800 NE Eighth, Suite 819
Bellevue, WA 98005
206-455-8080
Telex 328945

"Turn-Key" CP/M Systems

James J. Frantz

System power-up and loading procedures are almost too simple after implementing this technique. Perfect for small business (and home) applications.

The marriage of BASIC and Digital Research's CP/M has provided a powerful software team to the developers of business software. The hobbyist and the home computer game lover can also enjoy the result of this popular team. But, as the development of business systems reaches out to more and more users; ease of operation becomes more important.

As the development of business systems reaches out to more and more users, ease of operation becomes more important.

To bring a system up under CP/M after power-on is a multi-phased process and is not always easily performed by someone not accustomed to computer systems. As currently implemented, CP/M must first be loaded into memory from the diskette. This is normally very simple and involves inserting the diskette into the drive and depressing the "RESET" button. Once CP/M gets control, it responds with the "A>" prompt. The user must then type in the proper command. For most versions of BASIC the user must type a multi-word command in order to cause CP/M to load the BASIC which in turn will load and execute the desired program. In contrast to this elaborate procedure, most "big" systems immediately display a menu from which the user can select the desired program by entering the menu number. This minimizes, if not eliminates, the chance of typing errors. Why can't this be done with CP/M?

Well, it can. This article will describe how to make version 1.4 of CP/M start executing a program immediately after "RESET" is pressed. A menu program will also be provided that shows how this feature can be implemented with a game diskette. This technique was developed to allow my young daughter and her friends to select and run their choice of game from my collection of BASIC and machine language games without adult intervention. This same technique is even more suitable to business applications, especially dedicated systems.

CP/M Fundamentals

To understand how this works, the organization and operation of CP/M must be considered. CP/M is loaded into the top of existing memory from the diskette. There are various schemes used by vendors of disk systems which offer CP/M to accomplish this, but in every case CP/M begins execution after being loaded by entering at it's base. The base of CP/M is 2900H + b, where b is the bias determined by the amount of memory in the system. In a 16K system this bias is zero and a 32K system has a bias of 4000H. Starting at the base, CP/M is arranged as shown in Figure 1. The location of the input, or command buffer, and the storage location for the pointer to the command string are important in implementing automatic startup of a program.

Notice the zero byte at location seven. This zero tells CP/M that the command buffer is empty, i.e., there are zero characters in the buffer. The copyright notice which appears after the zero byte is over-written by whatever the user types after the "A>" appears on the screen. If this location contained something other than zero, CP/M would think that a command had already been typed, skip printing the prompt and begin processing the contents of the command buffer. By modifying the contents of the command buffer to contain a program name and changing the buffer length

Location 2900H+b+x (x)	Contents (Hex)	Description	Remarks
0 to 2	C3 55 zz*	JMP Instruction	Normal Entry Point
3 to 5	C3 51 zz*	JMP Instruction	Alternate Entry
6	7F	Max. Length of command buffer	Max. number of input characters allowed
7	00	Length of command string in buffer	Normally zero
8 to 23	20	ASCII blanks	
24 to 61	various	Copyright Notice	
62 to 135	00	Remainder of the command buffer	Initially Empty
136 & 137	00 (base)	Scan Pointer Storage	Points location 8

*zz = 2900H + b + 0300H

Figure 1

James Frantz, 94-265 Hiwahiwa Pl., Waipahn, HI 96797.

Turn-Key, con't...

byte to non-zero, CP/M can be made to execute that program every time CP/M is loaded just as if the user had typed it.

Also notice the storage location of the scan pointer. Initially this location contains the address of the beginning of the command buffer (base + 8).

This technique was developed to allow my young daughter and her friends to select and run their choice of game from my collection of BASIC and machine language games without adult intervention.

This pointer is updated as CP/M scans the command buffer during processing. After processing the command, the pointer is again stored at this location for possible continuation. This will become important when the need to start processing from the beginning of the command buffer is desired. Use of this storage location will be demonstrated in the MENU program described later.

Modifying CP/M

The procedure to modify CP/M is straightforward and can be easily accomplished with either SID or DDT. DDT is normally provided with CP/M when purchased, so this program will be used to make the modifications. First, be sure that a spare diskette is available to receive the modified CP/M. Next, a copy of CP/M must be made so DDT can bring it into memory. This is accomplished using SYSGEN, a program that is also distributed with CP/M. Run the SYSGEN program as follows:

SYSGEN start the SYSGEN program
SYSGEN VERSION 1.4 SYSGEN sign on message
SOURCE DRIVE NAME (OR RETURN TO SKIP) A Type "A"
SOURCE ON A, THEN TYPE RETURN

At this point, be sure the diskette in drive A contains the CP/M to be used, then type a "RETURN." The program should respond with:

FUNCTION COMPLETE CP/M is now in memory
DESTINATION DRIVE NAME (OR RETURN TO REBOOT)

The system will reboot at this point leaving a complete copy of CP/M in memory starting at 0980H and ending at 207FH. This copy of CP/M must be saved on the diskette. The save operation can be accomplished by typing:

SAVE 32 CPM.COM Save 32 pages of memory

```
D0980
0980 C3 55 6C C3 51 6C 7F 00 20 20 20 20 20 20 20 20  .Ul.Ql..
0990 20 20 20 20 20 20 20 20 43 4F 50 59 52 49 47 48          COPYRIGH
09A0 54 20 28 43 29 20 31 39 37 38 2C 20 44 49 47 49  T (C) 1978, DIGI
09B0 54 41 4C 20 52 45 53 45 41 52 43 48 20 20 00 00  TAL RESEARCH  ..
09C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
09D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
09E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
09F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0A00 00 00 00 00 00 00 00 00 03 69 00 00 5F 0E 02 C3  .........i._...
0A10 05 00 C5 CD 8C 69 C1 C9 3E 0D CD 8C 69 3E 0A C3  .....i..>...i>..
0A20 8C 69 C5 CD 98 69 E1 7E B7 C8 23 E5 CD 8C 69 E1  .i...i.~..#...i.
0A30 C3 A7 69 0E 0D C3 05 00 5F 0E 0E C3 05 00 0E 0F  ..i....._.......
```

Figure 3

Now that a copy of CP/M exists as a disk file, DDT can load this file for modification by typing:

DDT CPM.COM Load DDT which loads CPM.COM

DDT should respond with:

NEXT PC
2100 0100

CP/M is now back in memory starting at 0980H. Using the "D" command, type D0980[cr]. The display should look like Figure 2. Notice the two JMP instructions, followed by a 7F, then the zero byte. The zero byte will be replaced by the length of the program name, although any non-zero byte will work. After this length byte, enter the name of the program that is to automatically get control. This must be a disk file of type ".COM" in order to be executable. In the example, the program name is "MENU.COM," so "MENU" is entered beginning at location 0988H. The easiest way to enter these modifications is with the "S" command of DDT. Type "S0987[cr]" and DDT will respond with:

0987 00 - DDT waits for user entry

"MENU" is four characters in length, so enter "04" at location 0987H. DDT will respond with the next address. Enter the hexadecimal value for the each letter of the command name followed by a "RETURN." Repeat the process until the complete name is entered. Finally, and this is very important, enter a "00" immediately after the program name. After completing these entries, exit the "S" command of DDT by typing ".[cr]" and then perform another "D0980[cr]." The data displayed should now appear as shown in Figure 3.

```
D0980
0980 C3 55 6C C3 51 6C 7F 04 4D 45 4E 55 00 20 20 20  .Ul.Ql..MENU.
0990 20 20 20 20 20 20 20 20 43 4F 50 59 52 49 47 48          COPYRIGH
09A0 54 20 28 43 29 20 31 39 37 38 2C 20 44 49 47 49  T (C) 1978, DIGI
09B0 54 41 4C 20 52 45 53 45 41 52 43 48 20 20 00 00  TAL RESEARCH  ..
09C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
09D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
09E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
09F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0A00 00 00 00 00 00 00 00 00 03 69 00 00 5F 0E 02 C3  .........i._...
0A10 05 00 C5 CD 8C 69 C1 C9 3E 0D CD 8C 69 3E 0A C3  .....i..>...i>..
0A20 8C 69 C5 CD 98 69 E1 7E B7 C8 23 E5 CD 8C 69 E1  .i...i.~..#...i.
0A30 C3 A7 69 0E 0D C3 05 00 5F 0E 0E C3 05 00 0E 0F  ..i....._.......
```

Figure 2

When the copy of CP/M in memory has been correctly modified as described, exit DDT by typing: "G0[cr]." Without doing any intervening operation which might destroy the memory image of the modified CP/M, save this copy by typing:

SAVE 32 AUTOCPM.COM save 32 pages of memory

Then, using SYSGEN again, construct a diskette with the new system as follows:

SYSGEN start the SYSGEN program
SYSGEN VERSION 1.4 SYSGEN sign on message
SOURCE DRIVE NAME (OR RETURN TO SKIP) type "RETURN"
DESTINATION DRIVE NAME (OR RETURN TO REBOOT)A
DESTINATION ON A, THEN TYPE RETURN

Remove the diskette from drive A and insert the spare diskette which was previously prepared to receive the copy of the newly modified CP/M. When ready, type "RETURN." SYSGEN responds:

FUNCTION COMPLETE CP/M is now on diskette
DESTINATION DRIVE NAME (OR RETURN TO REBOOT)

Remove the new diskette and replace the original diskette. Type "RETURN" to reboot the system. The new diskette should be labeled and safely set aside until the MENU program has been prepared.

Turn-Key, con't...

Menu Program In BASIC

At this point it should be pointed out that some versions of BASIC which operate under CP/M allow entering a command in the form:

A BASIC MENU.BAA

where BASIC is a disk file of type ".COM." Entering a command in this form causes CP/M to load and execute BASIC.COM, which in turn will load and execute MENU.BAA. (Note that the file type ".BAA" is optional in some versions - check your User's Manual.) CBASIC and a version of MICROSOFT BASIC distributed by TEI, Inc. are known to perform in this way. If the menu scheme is to be implemented in a BASIC of this type, or if it is desired to have a specific BASIC program begin execution immediately after power-up, this can be done by entering the entire command string, exactly as it would be typed, in the command buffer. Be sure to enter a non-zero character count in location 0987H. Most importantly, the command string must be followed by a 00. If the command string is so long as to overwrite the copyright notice, move the copyright notice to after the 00 byte following the command string. All the space up to and including 0A07H can be used, but be absolutely sure that no modifications occur above this location.

Conditional assembly was used to allow both machine language programs (.COM files) and BASIC programs to be "menu-ized."

For those readers who use BASIC exclusively, the only work remaining is to transfer BASIC.COM and the desired startup program to the diskette with the CP/M which has been modified for automatic execution. If, however, the use of machine language programs is desired, or the version of BASIC doesn't have the facilities to load programs under the control of another BASIC program, the MENU program to be described might be the solution. In my case, new games were frequently being added to the game diskette, and many of the games were written in machine language. This program was developed to allow menu-ization of either BASIC games or machine language programs.

Assembly-Language Menu

The menu program in Listing A is written in 8080 machine language and is fully compatible with the Z-80. A fully commented listing is provided. The program has six major parts: 1) search and sort; 2) assign menu numbers; 3) compute column offsets; 4) display the menu in four columns; 5) input the user's menu selection; and 6) load and execute the selected program.

The search and sort loop uses the built-in capability of CP/M to search the diskette directory for files which match a specified pattern name. The pattern is selected so only the desired "type" of files are found. This is done by using the "?" which forces any character in the corresponding position to be a match. For example, if all files of type ".BAS" are wanted, a pattern of "????????BAS" would be specified. The desired pattern is set up in what is called a "File Control Block" or FCB for short. The standard convention for "calling" CP/M routines is to put the memory address of the FCB in the [DE] register pair, and the command number in register [C]. A call is made to the standard CP/M entry point which is memory location 0005. CP/M returns the directory "address" in the accumulator. Since 64 file names are allowed, this address is simply the sequential position in the diskette directory (0-63). If no file name matches the specified pattern, a FFH is returned. The "Search First" command is used to find the first occurrence of a match, and the "Search Next" command is used to find all subsequent file names which match the pattern.

As each file name is found, the directory address is converted to a memory address. Since all disk reads are performed in 128 byte blocks starting at the default address of 0080H, four directory entries are loaded into memory. Thus, the file names will be 32 bytes apart and only the two least significant bits of the directory address are needed. The file name is then alphabetically compared to the previous file names and inserted into its proper place in the "Directory Table" (labeled DIR$TABLE in the listing).

The second major part of the program assigns the menu numbers to each file name in the Directory Table. Notice that space was allocated in the Table for the menu number. This arrangement greatly facilitates the display of the menu in column form. The CP/M "print buffer" command requires a "$" as a termination character, so this is also inserted in the directory table to make printing the table entries easier.

The third major part of the program figures out the column arrangement based on the number of files to be displayed. The files are listed alphabetically from top to bottom in four columns. The algorithm seems complicated, but was devised so that each column would have the same number of file names with the extras being added from left to right.

The fourth part of the program displays the Directory Table on the user's console using the offsets computed in part three. Extra line feeds are added to completely fill a 16 line video display terminal. This is easily changed to accommodate 24 line displays, or can be deleted as desired.

The fifth part of the program displays instructions to the user and waits for the menu number to be entered. Incorrect entries force a redisplay of the menu. This portion of the program makes use of still another CP/M capability - buffered input. The [DE] register pairs are setup to point to a section of memory to be used to receive the input text. The first byte of this memory must contain the maximum length of the buffer area, and the second byte will be set by CP/M to the actual count of characters entered.

The last part of the program converts the menu number entered by the user into the program name. The ASCII representation which was entered from the console keyboard is converted to binary. This is then converted to the memory address of the corresponding file name within the Directory Table. Once the correct file name is pointed, the proper command string is positioned in the CP/M command buffer. Notice how conditional assembly was used to allow both machine language programs (.COM files) and BASIC programs to be "menu-ized." In the first case, only the file name followed by a zero byte is placed in the command buffer. In the case of BASIC, the name of the BASIC as a .COM file is followed by the selected program name. Some

The instructions to the operator would be reduced to explaining how to turn on the computer, how to properly insert the diskette, and the procedure for pressing the RESET button.

Turn-Key, con't...

versions of BASIC, such as CBASIC, assume a file type, while other versions require that the file type be specified in the command line. The program allows either case to be handled.

After the menu number is converted to the correct address within the Directory Table, the location of the CP/M command buffer must be found. The buffer is known to start at the base + 7 (refer to Figure 1). The base is a known distance below the address stored at location 0006H. The base is needed since this is the entry point which will cause CP/M to process the command line which the program has constructed. So, by doing some simple arithmetic, the base of CP/M and the location of the command buffer can be found. The program does these calculations and the desired command string is properly positioned. There is one remaining problem which the program must solve. When first executed after initial load, CP/M detected the prepositioned command which was installed by the modification to CP/M. During the processing of this command, the scan pointer was moved to the end of this command and is no longer in the correct position to scan the new command line built by the program. Fortunately, the scan pointer is stored in a known location immediately after the end of the command buffer. Since the command buffer location is known, so is the storage location of the scan pointer. The program resets the scan pointer to the beginning of the command buffer and CP/M is ready to process the command.

Summary

This article has explained how to make CP/M execute a program upon power up by prepositioning the command in the command buffer. The requirement to reset the scan pointer to force CP/M to again process the command buffer was also explained. The producers of dedicated system software can take advantage of this capability and offer a system which is simpler to operate. The instructions to the operator would be reduced to explaining how to turn on the computer, how to properly insert the diskette, and the procedure for pressing the RESET button. Any additional instructions could be displayed by the program automatically executed. A true "turn-key" system? Perhaps not, but how much closer can you get without Read Only Memory? Happy Computing! □

```
;                       MENU PROGRAM
;                           FOR
;              MACHINE LANGUAGE OR BASIC PROGRAMS
;
;                           BY
;                      JAMES J. FRANTZ
;                       MAY 31, 1979
;
;
;THIS PROGRAM IS DESIGNED TO BE AUTOMATICALLY EXECUTED
;BY CP/M IMMEDIATELY AFTER A COLD (OR WARM) BOOT. THIS
;PROGRAM THEN DISPLAYS THE CONTENTS OF THE DISK IN A
;MENU FASHION. ALL FILES OF SPECIFIED TYPE ARE SORTED AND
;DISPLAYED ALPHABETICALLY IN FOUR COLUMNS. THE USER THEN
;SELECTS THE DESIRED PROGRAM BY IT'S MENU NUMBER. THE
;SELECTED PROGRAM IS THEN RUN.
;
0100            ORG     0100H
;
;
FFFF =  BASIC$PROG      EQU     -1      ;SET TO 0 FOR MACHINE
                                        ;LANGUAGE GAMES
;
;
;FIRST THE CP/M 'SEARCH' COMMAND IS USED TO FIND THE
;FIRST FILE OF THE SPECIFIED TYPE. THE POINTER TO THE FILE
;CONTROL BLOCK IS PUT IN <DE>, AND THE COMMAND NUMBER IS
;PUT IN <C>. THE FILE CONTROL BLOCK IS PRE-CONSTRUCTED TO
;THE FORM '????????XXX0'. THE 'XXX' IS THE SPECIFIED FILE
;'TYPE', AND THE '?' FORCE A MATCH TO ANY FILE NAME OF THAT
;FILE TYPE.
;
0100 311A706            LXI     SP,STACK$AREA   ;SET UP A STACK
0103 0E11               MVI     C,17            ;'SEARCH FIRST' COMMAND
;
;THIS NEXT ROUTINE SORTS THE FILE NAMES AS THEY ARE FOUND
;ON THE DISK DIRECTORY. A NAME IS READ FROM THE DISK AND
;ITS LOCATION IS FOUND IN THE DIRECTORY TABLE BY COMPARING
;ALPHABETICALLY.
;
SORT$LOOP:
0105 11FC02             LXI     D,SRCH$FCB      ;POINT FILE CONTROL BLOCK
0108 CD0500             CALL    BDOS            ;USE CP/M ENTRY POINT
;
;CP/M RETURNS THE DISK ADDRESS OF THE NEXT MATCH IN <A>.
;THIS IS A VALUE BETWEEN 0 AND 64, OR -1 IF NO MATCH WAS
;FOUND. TEST FOR -1 AND QUIT WHEN NO MORE FILES OF THE
;SPECIFIED FILE 'TYPE' ARE FOUND ON THE DISK. THE DISK
;ADDRESS IS CONVERTED TO A POINTER TO THE FILE NAME
;WITHIN THE SECTOR BY MULTIPLYING BY 32 AND ADDING THE
;BASE ADDRESS OF THE SECTOR.
;
010B B7                 ORA     A               ;TEST FOR -1
010C FA6A01             JM      ASSIGN$MENU$NBR ;PRINT EMPTY MENU
010F 0F                 RRC                     ;THIS IS THE SAME AS
0110 0F                 RRC                     ;5 "ADD A'S"
0111 0F                 RRC
0112 E660               ANI     60H             ;MASK CORRECT BITS
0114 C680               ADI     80H             ;ADD BASE ADDRESS(0080H)
0116 5F                 MOV     E,A             ;PUT POINTER IN <DE>
0117 1600               MVI     D,0             ;AS 16 BIT VALUE
0119 210903             LXI     H,DIRTABLE      ;POINT START OF TABLE OF
                                                ;SORTED NAMES
011C 13                 INX     D               ;POINT PAST ERASE FIELD
;
COMPARE$LOOP:
011D D5                 PUSH    D               ;SAVE POINTER TO NEXT
                                                ;ENTRY FROM DISK DIRECTORY
011E 0E08               MVI     C,8             ;LENGTH OF COMPARE
0120 E5                 PUSH    H               ;SAVE POINTER TO TABLE
COMPARE1:
0121 1A                 LDAX    D               ;GET TRIAL NAME CHAR
0122 BE                 CMP     M               ;MATCH?
0123 C22C01             JNZ     END$COMPARE     ;IF NOT, TRY NEXT ENTRY
0126 23                 INX     H               ;ADVANCE POINTERS
0127 13                 INX     D
0128 0D                 DCR     C               ;ONE LESS CHAR TO COMPARE
0129 C22101             JNZ     COMPARE1        ;KEEP TESTING
END$COMPARE:
012C C1                 POP     B               ;RESTORE TABLE POINTER
012D DA3801             JC      INSERT$NAME     ;DIRECTORY NAME GOES IN
                                                ;FRONT OF CURRENT TABLE
                                                ;ENTRY IF LOWER(CY=1)
0130 210E00             LXI     H,14            ;LENGTH OF TABLE ENTRY
0133 09                 DAD     B               ;<HL> TO NEXT TABLE ENTRY
0134 D1                 POP     D               ;RECOVER TRIAL NAME POINT
0135 C31D01             JMP     COMPARE$LOOP    ;LOOP AGAIN
;
;THIS NEXT PORTION MAKES ROOM IN THE DIRECTORY TABLE FOR
;THE NEW ENTRY BY MOVING ALL ALPHABETICALLY HIGHER NAMES
;UPWARD IN MEMORY.
;
INSERT$NAME:
0138 21F902             LXI     H,FILE$COUNT    ;COUNT THE NUMBER OF FILES
013B 34                 INR     M               ;TO BE DISPLAYED
013C 2AF702             LHLD    END$OF$TABLE    ;GET POINTER TO TABLE END
013F EB                 XCHG
0140 210E00             LXI     H,14            ;DISTANCE TO MOVE
0143 19                 DAD     D               ;<HL> POINT DESTINATION
0144 22F702             SHLD    END$OF$TABLE    ;SAVE THE NEW END OF TABLE
0147 23                 INX     H
0148 13                 INX     D
;
MOVE$UP:
0149 1B                 DCX     D
014A 2B                 DCX     H
014B 1A                 LDAX    D               ;GET BYTE TO MOVE
014C 77                 MOV     M,A             ;PUT IN NEW SPOT
014D 79                 MOV     A,C             ;TEST FOR DONE
014E B8                 CMP     E               ;<BC> = <DE>?
014F C24901             JNZ     MOVE$UP
0152 78                 MOV     A,B
0153 BA                 CMP     D
0154 C24901             JNZ     MOVE$UP
0157 E1                 POP     H               ;RECOVER POINTER
0158 0E08               MVI     C,8
015A CD7D02             CALL    BLOCK$MOVE      ;INSERT NAME IN TABLE
;
;THE MENU NUMBER FIELD IS INSERTED IN THE DIRECTORY TABLE
;AT THIS POINT BUT THE ACTUAL MENU NUMBER WILL BE ASSIGNED
;AFTER ALL FILES ARE SORTED.
;
015D 21ED02             LXI     H,MENU$BUFF     ;POINT MENU NUMBER BLOCK
0160 0E06               MVI     C,6             ;LENGTH OF MOVE
0162 CD7D02             CALL    BLOCK$MOVE      ;INSERT TEXT IN TABLE
;
;THE COMMAND NUMBER FOR SUBSEQUENT SEARCHES OF THE DISK
;DIRECTORY MUST BE ALTERED TO CAUSE CP/M TO SEARCH FROM
```

Turn-Key, con't...

```
                        ;WHERE IT LEFT OFF.
                        ;
0165 0E12               MVI     C,18            ;'SEARCH NEXT' COMMAND
0167 C38501             JMP     SORT$LOOP
                        ;
                        ;
                        ;THIS IS THE SECOND MAJOR PORTION OF THE PROGRAM. AT THIS
                        ;POINT ALL FILES HAVE BEEN INSERTED IN THE DIRECTORY TABLE
                        ;IN ALPHABETICAL ORDER. NOW THE MENU NUMBERS ARE ASSIGNED
                        ;AND INSERTED IN THE PROPER PLACE IN PREPARATION FOR
                        ;DISPLAY ON THE TERMINAL.
                        ;
                        ASSIGN$MENU$NBR:
016A 3AF902             LDA     FILE$COUNT
016D 47                 MOV     B,A             ;SAVE IN <B>
016E F5                 PUSH    PSW             ;AND ON STACK
016F 0E00               MVI     C,0             ;INITIAL FILE NUMBER
0171 211403             LXI     H,DIRTABLE+11   ;POINT FIRST FILE NUMBER
0174 110D00             LXI     D,13            ;OFFSET TO OTHER NUMBERS
                        NUMBER$FILES:
0177 79                 MOV     A,C             ;PUT FILE NUMBER IN <A>
0178 C601               ADI     1               ;INCREMENT
017A 27                 DAA                     ;DECIMAL CONVERT
017B 4F                 MOV     C,A             ;RESAVE IN <C>
017C 0F                 RRC                     ;GET TENS DIGIT INTO
017D 0F                 RRC                     ;PROPER PLACE
017E 0F                 RRC
017F 0F                 RRC
0180 E60F               ANI     0FH             ;AND MASK
0182 CA8701             JZ      USE$BLANK       ;SUPPRESS LEADING ZERO BY
0185 C610               ADI     10H             ;ADD EITHER 20H(ASCII ' ')
                        USE$BLANK:
0187 C620               ADI     ' '             ;OR 20H + 10H FOR NUMERAL
0189 77                 MOV     M,A             ;PUT IN TEXT STREAM
018A 79                 MOV     A,C             ;GET UNITS PORTION
018B E60F               ANI     0FH             ;MASK OFF TENS PORTION
018D C630               ADI     '0'             ;CONVERT TO ASCII
018F 23                 INX     H
0190 77                 MOV     M,A
0191 19                 DAD     D               ;REPEAT UNTIL ALL FILES
0192 05                 DCR     B               ;ARE SEQUENTIALLY NUMBERED
0193 C27701             JNZ     NUMBER$FILES

0196 F1                 POP     PSW             ;GET FILE$COUNT FROM STACK
0197 F5                 PUSH    PSW             ;AND SAVE AGAIN FOR LATER
                        ;
                        ;THIS ALGORYTHM ENSURES THE COLUMNS ARE AS EVEN IN LENGTH
                        ;AS POSSIBLE. DON'T WORRY, IT WORKS.
                        ;
0198 C603               ADI     NBR$COL - 1
019A 06FF               MVI     B,-1            ;<B> ACCUMULATES QUOTIENT
                                                ;SO SET TO -1 FOR AT LEAST
                                                ;1 PASS THRU GIVES 0
019C 04                 DIVX    INR     B
019D D604               SUI     NBR$COL         ;DIVIDE (FILE$COUNT+3) BY
                                                ;FOUR TO GET OFFSET1
019F F29C01             JP      DIVX
01A2 C604               ADI     NBR$COL         ;SUBTRACTED ONCE TOO MUCH
                                                ;SO ADD IT BACK ON
01A4 21F402             LXI     H,OFFSET1
01A7 78                 MOV     M,B             ;INSERT OFFSET1 INTO TABLE
01A8 23                 INX     H               ;POINT OFFSET2 LOCATION
01A9 C2AD01             JNZ     SETOFFSET2      ;SAME AS OFFSET1 IF NON-
01AC 05                 DCR     B               ;ZERO REMAINDER
                                                ;ELSE OFFSET2=OFFSET1-1
                        SETOFFSET2:
01AD 70                 MOV     M,B             ;PUT OFFSET2 IN TABLE
01AE 23                 INX     H               ;POINT OFFSET FOR COL 3
01AF 3D                 DCR     A               ;TEST FOR REMAINER OF 1
0180 C2B401             JNZ     SETOFFSET3      ;IF REMAINDER <> 1, USE
                                                ;OFFSET3=OFFSET2
0183 05                 DCR     B               ;ELSE OFFSET3=OFFSET2-1
                        SETOFFSET3:
0184 70                 MOV     M,B             ;OFFSET TO COLUMN 4
                        ;
                        ;NOW THAT THE OFFSETS FOR THE COLUMNS HAVE BEEN DETERMINED
                        ;THE ACTUAL PRINT OUT CAN BE MADE.
                        ;
                        REPRINT:
0185 F1                 POP     PSW             ;RECOVER FILE COUNT
                        REPRINT1:
0186 F5                 PUSH    PSW             ;SAVE AGAIN FOR LATER USE
0187 32F902             STA     FILE$COUNT      ;SAVE FOR COUNTING
018A 3E0C               MVI     A,SCREEN$HGT    ;SET FOR VIDEO DISPLAY SIZE
018C 32FB02             STA     LINE$COUNT
018F 11A802             LXI     D,HEADING
01C2 0E09               MVI     C,9             ;BUFFER PRINT COMMAND
01C4 CD0500             CALL    BDOS            ;CP/M PRINTS HEADING
                        ;
01C7 21FB02             LXI     H,DIR$TABLE - 14 ;POINT DUMMY 0TH ENTRY
                        ;
                        PRINT$LINE:
01CA E5                 PUSH    H               ;SAVE BASE ADDRESS
01CB 11F302             LXI     D,OFFSET0       ;POINT OFFSET TABLE
01CE 3E04               MVI     A,NBR$COL       ;4 COLUMNS PER LINE
                        PRINT$NAME:
01D0 32FA02             STA     COLUMN$CNT      ;SAVE COUNT OF COLUMNS
01D3 E5                 PUSH    H               ;SAVE CURRENT NAME POINTER
01D4 D5                 PUSH    D               ;SAVE OFFSET TABLE POINTER
                        ;
01D5 11EA02             LXI     D,DOUBL$SPACE   ;PRINT 2 BLANKS
01D8 0E09               MVI     C,9             ;'PRINT BUFFER' COMMAND
01DA CD0500             CALL    BDOS            ;USE CP/M FACILITY
01DD D1                 POP     D               ;GET OFFSET TABLE POINTER
01DE E1                 POP     H               ;GET NAME POINTER
01DF 1A                 LDAX    D               ;GET OFFSET VALUE
                        ;
                        ;THE OFFSET VALUE IS THE NUMBER OF FILE NAMES FROM THE
                        ;CURRENT NAME POINTER TO MOVE THE PRINT POINTER.
                        ;
01E0 010E00             LXI     B,14            ;EACH NAME IS 14 LONG
                        MULT$14:
01E3 09                 DAD     B               ;ADD 14 FOR EACH OFFSET
01E4 3D                 DCR     A               ;UNTIL OFFSET = 0
01E5 C2E301             JNZ     MULT$14
01E8 E5                 PUSH    H               ;SAVE NEW NAME POINTER
01E9 D5                 PUSH    D               ;SAVE OFFSET POINTER
01EA EB                 XCHG                    ;POINT NAME TO PRINT W/<DE>
01EB 0E09               MVI     C,9             ;PRINT BUFFER
01ED CD0500             CALL    BDOS            ;PRINT THE FILE NAME AND
                                                ;ITS MENU NUMBER
                        ;
                        TEST$FINISH:
01F0 21F902             LXI     H,FILE$COUNT    ;SEE IF DONE PRINTING
01F3 35                 DCR     M               ;BY TESTING COUNT OF FILES
01F4 D1                 POP     D               ;GET OFFSET POINTER
01F5 E1                 POP     H               ;GET POINTER TO LAST NAME
```

```
01F6 CA0C02             JZ      FINISH          ;NO MORE TO PRINT.
01F9 13                 INX     D               ;ADVANCE OFFSET POINTER
01FA 3AFA02             LDA     COLUMN$CNT
01FD 3D                 DCR     A               ;SEE IF COLUMNS LEFT = 0
01FE C2D001             JNZ     PRINT$NAME      ;PRINT ANOTHER SAME LINE.
0201 CD9802             CALL    CRLF            ;MOVE TO NEXT LINE
0204 E1                 POP     H               ;GET BASE OF PREVIOUS LINE
0205 110E00             LXI     D,14            ;ADD OFFSET
0208 19                 DAD     D
0209 C3CA01             JMP     PRINT$LINE
                ;
                ;THE FILE NAMES AND THEIR MENU NUMBERS HAVE BEEN PRINTED.
                ;THIS NEXT LOOP OUTPUTS SUFFICIENT LINE FEEDS TO PUT THE
                ;HEADING AT THE TOP OF A 16 LINE VIDEO DISPLAY(THEREBY
                ;CLEARING THE SCREEN), AND PUTS THE REQUEST FOR USER
                ;SELECTION AT THE BOTTOM OF THE SCREEN.
                ;
                ;
                FINISH:
020C E1                 POP     H               ;UNJUNK STACK
                LF$LOOP:
020D CD9802             CALL    CRLF
0210 F20D02             JP      LF$LOOP         ;OMIT THIS LINE IF DESIRED
0213 11C402             LXI     D,PROMPT        ;POINT INSTRUCTION MSG.
0216 0E09               MVI     C,9
0218 CD0500             CALL    BDOS            ;AGAIN CP/M PRINTS MESSAGE
021B 11A706             LXI     D,INPUT$BUFF
021E 3E0A               MVI     A,10            ;TEN CHARACTERS MAX
0220 12                 STAX    D
0221 0E0A               MVI     C,10            ;'READ BUFFER' COMMAND
0223 CD0500             CALL    BDOS
                ;
                ;ON RETURN FROM BDOS LINE INPUT FUNCTION, THE DIGITS
                ;TYPED BY THE USER ARE IN THE BUFFER AT INPUT$BUFF+2.
                ;CONVERT TO BINARY.
                ;
0226 21A806             LXI     H,INPUT$BUFF+1  ;POINT CHAR COUNT
0229 7E                 MOV     A,M             ;GET IT AND SEE IF > 2
022A FE03               CPI     3
022C D28501             JNC     REPRINT         ;REPRINT THE MENU
022F 4F                 MOV     C,A             ;COUNT OF DIGITS TO <C>
0230 0600               MVI     B,0             ;ZERO <B>
                ;
                GET$MENU$NBR:
0232 23                 INX     H               ;POINT ASCII DIGIT
0233 7E                 MOV     A,M             ;GET IT
0234 CD8602             CALL    ASCII$CONVERT   ;CONVERT TO BINARY
0237 DA8501             JC      REPRINT         ;RE-DISPLAY ON ERROR
023A 0D                 DCR     C
023B C23202             JNZ     GET$MENU$NBR
                ;
                ;<B> HAS THE MENU NUMBER. TO BE SURE THIS IS A LEGAL MENU
                ;REQUEST, COMPARE WITH FILE$COUNT (STILL ON STACK).
                ;
023E F1                 POP     PSW             ;RECOVER FILE COUNT
023F B8                 CMP     B               ;FILE$COUNT-REQUEST NBR.
0240 DA8601             JC      REPRINT1        ;RE-DISPLAY MENU IF ILLEGAL.
                ;
                ;
0243 110E00             LXI     D,14            ;INCREMENT BETWEEN NAMES
0246 21FB02             LXI     H,DIR$TABLE - 14 ;POINT DUMMY 0TH ENTRY
                FIND$NAME:
0249 19                 DAD     D               ;ADD OFFSET <B> TIMES
024A 05                 DCR     B

024B C24902             JNZ     FIND$NAME
                ;
                ;AT THIS POINT <HL> POINTS THE SELECTED FILE NAME. NOW FIND
                ;THE ADDRESS OF CP/M SO THE PROPER COMMAND NAME AND THE
                ;SELECTED FILE NAME CAN BE PUT INTO THE COMMAND BUFFER.
                ;
024E EB                 XCHG                    ;SAVE POINTER TO FILE NAME
024F 2A0600             LHLD    6               ;GET BDOS ENTRY POINT
0252 01FAF7             LXI     B,- CCP$LEN     ;OFFSET TO START OF CP/M
0255 09                 DAD     B
0256 E5                 PUSH    H               ;SAVE CP/M ENTRY POINT
                                                ;ON STACK FOR BRANCH.
0257 010800             LXI     B,8             ;OFFSET TO COMMAND BUFFER
025A 09                 DAD     B               ;<HL> POINTS PLACE TO PUT
                                                ;NAME OF .COM FILE TO BE
                                                ;EXECUTED.
025B D5                 PUSH    D               ;SAVE POINTER TO FILE NAME
025C EB                 XCHG                    ;<DE> POINTS COMMAND BUFF
                ;
                ;SINCE THE SCAN POINTER IS NOT RESET BY REENTRY INTO CP/M
                ;THE SCAN POINTER MUST BE RESET BY THIS PROGRAM. THE SCAN
                ;POINTER IS STORED BY CP/M AT THE END OF THE COMMAND
                ;BUFFER.
                ;
025D 218000             LXI     H,128           ;OFFSET TO END OF CMD BUFF
                                                ;WHERE POINTER IS STORED
0260 19                 DAD     D               ;<HL> POINTS STORAGE PLACE
0261 73                 MOV     M,E             ;UPDATE BUFFER POINTER TO
0262 23                 INX     H               ;THE START OF THE COMMAND
0263 72                 MOV     M,D             ;BUFF SO CP/M WILL READ.
                ;
                        IF      BASIC$PROG      ;CONDITIONAL ASSEMBLY
                                                ;FOR BASIC TO RUN THE
                                                ;SELECTED PROGRAM
0264 21BA02             LXI     H,COMMAND$NAME  ;POINT COMMAND NAME
0267 0E06               MVI     C,LEN$CMD$NAME  ;LENGTH OF COMMAND NAME
0269 CD7D02             CALL    BLOCK$MOVE
                ;
                        ENDIF
                ;
                ;
026C E1                 POP     H               ;POINT SELECTED FILE NAME
026D 0E08               MVI     C,8             ;LENGTH OF FILE NAME
026F CD7D02             CALL    BLOCK$MOVE
                ;
                ;*****************************************************
                ;IF THE VERSION OF BASIC TO BE USED DOES NOT REQUIRE THE
                ;FILE TYPE TO BE SPECIFIED IN THE COMMAND LINE, OMIT THE
                ;LINES BETWEEN THE ASTERICK LINES.
                ;
                        IF      BASIC$PROG      ;AGAIN FOR SOME VERSIONS
                                                ;OF BASIC WHICH REQUIRE
                                                ;THE FILE TYPE TO BE
                                                ;SPECIFIED.
0272 21C002             LXI     H,SPEC$TYPE     ;NEEDED BY SOME VERSIONS
                                                ;OF BASIC
0275 0E04               MVI     C,4             ;LENGTH OF 'TYPE' FIELD
0277 CD7D02             CALL    BLOCK$MOVE
                ;
                        ENDIF
                ;
                ;*****************************************************
                ;
```

```
                ;
027A AF         XRA     A               ;NEED A ZERO AT END
027B 12         STAX    D               ;OF COMMAND LINE
                ;
                ;THE ADDRESS OF CP/M IS ON THE STACK SO A SIMPLE RETURN
                ;WILL EXECUTE CP/M AND IN TURN EXECUTE `BASIC [FILENAME]'.
                ;
027C C9         RET
                ;
                ;SUBROUTINES:
                ;
                BLOCK$MOVE:
027D 7E         MOV     A,M
027E 12         STAX    D
027F 13         INX     D
0280 23         INX     H
0281 0D         DCR     C
0282 C27D02     JNZ     BLOCK$MOVE
0285 C9         RET
                ;
                ASCII$CONVERT:
0286 D630       SUI     '0'             ;SUBTRACT ASCII BIAS
0288 FE0A       CPI     9 + 1           ;BE SURE IT'S NUMERIC
028A 3F         CMC
028B D8         RC                      ;SET CY=1 IF ILLEGAL
028C 57         MOV     D,A             ;SAVE IN <D>
028D 78         MOV     A,B             ;GET PREVIOUS RESULT
028E 07         RLC                     ;MULTIPLY BY 2
028F 07         RLC                     ;THEN 4
0290 07         RLC                     ;THEN 8
0291 80         ADD     B               ;THEN 9
0292 D8         RC                      ;OUT OF BOUNDS
0293 80         ADD     B               ;THEN FINALLY BY 10
0294 D8         RC                      ;CY=1 ALWAYS MEANS ERROR
0295 82         ADD     D               ;ADD IN NEW RESULT
0296 47         MOV     B,A             ;SAVE IN (B)
0297 C9         RET
                ;
                CRLF:
0298 11A502     LXI     D,CRLFMSG
029B 0E09       MVI     C,9             ;BUFFER PRINT
029D CD0500     CALL    BDOS
02A0 21FB02     LXI     H,LINE$COUNT
02A3 35         DCR     M
02A4 C9         RET
                ;
02A5 0D0A24     CRLFMSG DB      0DH,0AH,'$'
                ;
02A8 0A0909092EHEADING DB       0AH,9,9,9,'.    MENU',0DH,0AH,0AH,'$'
                ;
                        IF      BASIC$PROG      ;CONDITIONAL ASSEMBLY FOR
                                                ;BASIC.COM + FILENAME
                ;
                COMMAND$NAME:
02BA 4241534943         DB      'BASIC '        ;INSERT THE NAME OF THE
                                                ;BASIC TO BE USED (E.G.
                                                ;`CRUN', `CRUN2', ETC.)
                ;
0006 =          LEN$CMD$NAME    EQU     $-COMMAND$NAME
                ;
                SPEC$TYPE:
```

```
02C0 2E424141           DB      '.BAA'          ;ONLY IF REQUIRED BY BASIC
                                                ;VERSION USED.
                ;
                        ENDIF
                ;
                PROMPT:
02C4 454E544552         DB      'ENTER MENU NUMBER AND PRESS `RETURN'' $'
                ;
                DOUBL$SPACE:
02EA 202024             DB      '  $'
                ;
                MENU$BUFF:
02ED 202D202030         DB      ' - 0$'
                ;
                ;
02F3 01         OFFSET0 DB      1
02F4 000000     OFFSET1 DB      0,0,0
                ;
                END$OF$TABLE:
02F7 0903               DW      DIRTABLE
                ;
                FILE$COUNT:
02F9 00                 DB      0
                COLUMN$CNT:
02FA 04                 DB      4
                LINE$COUNT:
02FB 00                 DB      0
                ;
                ;IN THE NEXT LINE INSERT THE FILE TYPE OF THE FILES TO BE
                ;DISPLAYED.
                ;
                        IF      BASIC$PROG
                ;
                SRCH$FCB:
02FC 003F3F3F3F         DB      0,'????????BAA',0
                ;
                        ELSE                    ;FOR MACHINE LANGUAGE
                ;
                SRCH$FCB:
                        DB      0,'????????COM',0       ;FOR MACHINE
                                                        ;LANGUAGE GAMES
                ;
                        ENDIF
                ;
                ;
                DIR$TABLE:
0309 FF                 DB      -1              ;FORCE FIRST FILE
                                                ;TO BE PUT HERE
                ;
06A7 =          STACK$AREA EQU  DIR$TABLE + 64*14 + 30
                ;
06A7 =          INPUT$BUFF EQU  STACK$AREA
                ;
                ;EQUATES:
                ;
0005 =          BDOS    EQU     5
0004 =          NBR$COL EQU     4
0806 =          CCP$LEN EQU     3106H-2900H
0010 =          SCREEN$SIZE EQU 16              ;CHANGE AS APPROPRIATE
                                                ;FOR VIDEO
000C =          SCREEN$HGT  EQU SCREEN$SIZE - 4
                ;
```

Turn-Key, con't...

```
                ;                       MENU PROGRAM
                ;                           FOR
                ;               MACHINE LANGUAGE OR BASIC PROGRAMS
                ;
                ;                           BY
                ;                       JAMES J. FRANTZ
                ;                       MAY 31, 1979
                ;
                ;
                ;THIS PROGRAM IS DESIGNED TO BE AUTOMATICALLY EXECUTED
                ;BY CP/M IMMEDIATELY AFTER A COLD (OR WARM) BOOT. THIS
                ;PROGRAM THEN DISPLAYS THE CONTENTS OF THE DISK IN A
                ;MENU FASHION. ALL FILES OF SPECIFIED TYPE ARE SORTED AND
                ;DISPLAYED ALPHABETICALLY IN FOUR COLUMNS. THE USER THEN
                ;SELECTS THE DESIRED PROGRAM BY IT'S MENU NUMBER. THE
                ;SELECTED PROGRAM IS THEN RUN.
                ;
0100                    ORG     0100H
                ;
                ;
FFFF =          BASIC$PROG      EQU     -1      ;SET TO 0 FOR MACHINE
                                                ;LANGUAGE GAMES
                ;
                ;
                ;FIRST THE CP/M `SEARCH' COMMAND IS USED TO FIND THE
                ;FIRST FILE OF THE SPECIFIED TYPE. THE POINTER TO THE FILE
                ;CONTROL BLOCK IS PUT IN <DE>, AND THE COMMAND NUMBER IS
                ;PUT IN <C>. THE FILE CONTROL BLOCK IS PRE-CONSTRUCTED TO
                ;THE FORM `????????XXX0'. THE `XXX' IS THE SPECIFIED FILE
                ;`TYPE', AND THE `?' FORCE A MATCH TO ANY FILE NAME OF THAT
                ;FILE TYPE.
                ;
0100 31A706             LXI     SP,STACK$AREA   ;SET UP A STACK
0103 0E11               MVI     C,17            ;`SEARCH FIRST' COMMAND
                ;
                ;THIS NEXT ROUTINE SORTS THE FILE NAMES AS THEY ARE FOUND
                ;ON THE DISK DIRECTORY. A NAME IS READ FROM THE DISK AND
                ;ITS LOCATION IS FOUND IN THE DIRECTORY TABLE BY COMPARING
                ;ALPHABETICALLY.
                ;
                SORT$LOOP:
0105 11FC02             LXI     D,SRCH$FCB      ;POINT FILE CONTROL BLOCK
0108 CD0500             CALL    BDOS            ;USE CP/M ENTRY POINT
                ;
                ;CP/M RETURNS THE DISK ADDRESS OF THE NEXT MATCH IN <A>.
                ;THIS IS A VALUE BETWEEN 0 AND 64, OR -1 IF NO MATCH WAS
                ;FOUND. TEST FOR -1 AND QUIT WHEN NO MORE FILES OF THE
                ;SPECIFIED FILE `TYPE' ARE FOUND ON THE DISK. THE DISK
                ;ADDRESS IS CONVERTED TO A POINTER TO THE FILE NAME
                ;WITHIN THE SECTOR BY MULTIPLYING BY 32 AND ADDING THE
                ;BASE ADDRESS OF THE SECTOR.
                ;
010B B7                 ORA     A               ;TEST FOR -1
010C FA6A01             JM      ASSIGN$MENU$NBR ;PRINT EMPTY MENU
010F 0F                 RRC                     ;THIS IS THE SAME AS
0110 0F                 RRC                     ;5 "ADD A'S"
0111 0F                 RRC
0112 E660               ANI     60H             ;MASK CORRECT BITS
0114 C680               ADI     80H             ;ADD BASE ADDRESS(0080H)
0116 5F                 MOV     E,A             ;PUT POINTER IN <DE>
0117 1600               MVI     D,0             ;AS 16 BIT VALUE
0119 210903             LXI     H,DIRTABLE      ;POINT START OF TABLE OF
                                                ;SORTED NAMES
011C 13                 INX     D               ;POINT PAST ERASE FIELD
                COMPARE$LOOP:
011D D5                 PUSH    D               ;SAVE POINTER TO NEXT
                                                ;ENTRY FROM DISK DIRECTORY
011E 0E08               MVI     C,8             ;LENGTH OF COMPARE
0120 E5                 PUSH    H               ;SAVE POINTER TO TABLE
                COMPARE1:
0121 1A                 LDAX    D               ;GET TRIAL NAME CHAR
0122 BE                 CMP     M               ;MATCH?
0123 C22C01             JNZ     END$COMPARE     ;IF NOT, TRY NEXT ENTRY
0126 23                 INX     H               ;ADVANCE POINTERS
0127 13                 INX     D
0128 0D                 DCR     C               ;ONE LESS CHAR TO COMPARE
0129 C22101             JNZ     COMPARE1        ;KEEP TESTING
                END$COMPARE:
012C C1                 POP     B               ;RESTORE TABLE POINTER
012D DA3801             JC      INSERT$NAME     ;DIRECTORY NAME GOES IN
                                                ;FRONT OF CURRENT TABLE
                                                ;ENTRY IF LOWER(CY=1)
0130 210E00             LXI     H,14            ;LENGTH OF TABLE ENTRY
0133 09                 DAD     B               ;<HL> TO NEXT TABLE ENTRY
0134 D1                 POP     D               ;RECOVER TRIAL NAME POINT
0135 C31D01             JMP     COMPARE$LOOP    ;LOOP AGAIN
                ;
                ;THIS NEXT PORTION MAKES ROOM IN THE DIRECTORY TABLE FOR
                ;THE NEW ENTRY BY MOVING ALL ALPHABETICALLY HIGHER NAMES
                ;UPWARD IN MEMORY.
                ;
                INSERT$NAME:
0138 21F902             LXI     H,FILE$COUNT    ;COUNT THE NUMBER OF FILES
013B 34                 INR     M               ;TO BE DISPLAYED
013C 2AF702             LHLD    END$OF$TABLE    ;GET POINTER TO TABLE END
013F EB                 XCHG
0140 210E00             LXI     H,14            ;DISTANCE TO MOVE
0143 19                 DAD     D               ;<HL> POINT DESTINATION
0144 22F702             SHLD    END$OF$TABLE    ;SAVE THE NEW END OF TABLE
0147 23                 INX     H
0148 13                 INX     D
                ;
                MOVE$UP:
0149 1B                 DCX     D
014A 2B                 DCX     H
014B 1A                 LDAX    D               ;GET BYTE TO MOVE
014C 77                 MOV     M,A             ;PUT IN NEW SPOT
014D 79                 MOV     A,C             ;TEST FOR DONE
014E BB                 CMP     E               ;<BC> = <DE>?
014F C24901             JNZ     MOVE$UP
0152 78                 MOV     A,B
0153 BA                 CMP     D
0154 C24901             JNZ     MOVE$UP
0157 E1                 POP     H               ;RECOVER POINTER
0158 0E08               MVI     C,8
015A CD7D02             CALL    BLOCK$MOVE      ;INSERT NAME IN TABLE
                ;
                ;THE MENU NUMBER FIELD IS INSERTED IN THE DIRECTORY TABLE
                ;AT THIS POINT BUT THE ACTUAL MENU NUMBER WILL BE ASSIGNED
                ;AFTER ALL FILES ARE SORTED.
                ;
0150 21ED02             LXI     H,MENU$BUFF     ;POINT MENU NUMBER BLOCK
0160 0E06               MVI     C,6             ;LENGTH OF MOVE
0162 CD7D02             CALL    BLOCK$MOVE      ;INSERT TEXT IN TABLE
                ;
                ;THE COMMAND NUMBER FOR SUBSEQUENT SEARCHES OF THE DISK
                ;DIRECTORY MUST BE ALTERED TO CAUSE CP/M TO SEARCH FROM
```


Turn-Key, con't...

```
                        ;WHERE IT LEFT OFF.
                        ;
0165 0E12               MVI     C,18            ;'SEARCH NEXT' COMMAND
0167 C38501             JMP     SORT$LOOP
                        ;
                        ;
                        ;THIS IS THE SECOND MAJOR PORTION OF THE PROGRAM. AT THIS
                        ;POINT ALL FILES HAVE BEEN INSERTED IN THE DIRECTORY TABLE
                        ;IN ALPHABETICAL ORDER. NOW THE MENU NUMBERS ARE ASSIGNED
                        ;AND INSERTED IN THE PROPER PLACE IN PREPARATION FOR
                        ;DISPLAY ON THE TERMINAL.
                        ;
                        ASSIGN$MENU$NBR:
016A 3AF902             LDA     FILE$COUNT
016D 47                 MOV     B,A             ;SAVE IN <B>
016E F5                 PUSH    PSW             ;AND ON STACK
016F 0E00               MVI     C,0             ;INITIAL FILE NUMBER
0171 211403             LXI     H,DIRTABLE+11   ;POINT FIRST FILE NUMBER
0174 110D00             LXI     D,13            ;OFFSET TO OTHER NUMBERS
                        NUMBER$FILES:
0177 79                 MOV     A,C             ;PUT FILE NUMBER IN <A>
0178 C601               ADI     1               ;INCREMENT
017A 27                 DAA                     ;DECIMAL CONVERT
017B 4F                 MOV     C,A             ;RESAVE IN <C>
017C 0F                 RRC                     ;GET TENS DIGIT INTO
017D 0F                 RRC                     ;PROPER PLACE
017E 0F                 RRC
017F 0F                 RRC
0180 E60F               ANI     0FH             ;AND MASK
0182 CA8701             JZ      USE$BLANK       ;SUPPRESS LEADING ZERO BY
0185 C610               ADI     10H             ;ADD EITHER 20H(ASCII ' ')
                        USE$BLANK:
0187 C620               ADI     ' '             ;OR 20H + 10H FOR NUMERAL
0189 77                 MOV     M,A             ;PUT IN TEXT STREAM
018A 79                 MOV     A,C             ;GET UNITS PORTION
018B E60F               ANI     0FH             ;MASK OFF TENS PORTION
018D C630               ADI     '0'             ;CONVERT TO ASCII
018F 23                 INX     H
0190 77                 MOV     M,A
0191 19                 DAD     D               ;REPEAT UNTIL ALL FILES
0192 05                 DCR     B               ;ARE SEQUENTIALLY NUMBERED
0193 C27701             JNZ     NUMBER$FILES
                        ;
0196 F1                 POP     PSW             ;GET FILE$COUNT FROM STACK
0197 F5                 PUSH    PSW             ;AND SAVE AGAIN FOR LATER
                        ;
                        ;THIS ALGORYTHM ENSURES THE COLUMNS ARE AS EVEN IN LENGTH
                        ;AS POSSIBLE. DON'T WORRY, IT WORKS.
                        ;
0198 C603               ADI     NBR$COL - 1
019A 06FF               MVI     B,-1            ;<B> ACCUMULATES QUOTIENT
                                                ;SO SET TO -1 FOR AT LEAST
                                                ;1 PASS THRU GIVES 0
019C 04                 DIVX    INR     B
019D D604                       SUI     NBR$COL ;DIVIDE (FILE$COUNT+3) BY
                                                ;FOUR TO GET OFFSET!
019F F29C01                     JP      DIVX
01A2 C604                       ADI     NBR$COL ;SUBTRACTED ONCE TOO MUCH
                                                ;SO ADD IT BACK ON
01A4 21F402             LXI     H,OFFSET1
01A7 70                 MOV     M,B             ;INSERT OFFSET1 INTO TABLE
01A8 23                 INX     H               ;POINT OFFSET2 LOCATION
01A9 C2AD01             JNZ     SETOFFSET2      ;SAME AS OFFSET1 IF NON-
01AC 05                                         ;ZERO REMAINDER
                        DCR     B               ;ELSE OFFSET2=OFFSET1-1
                        SETOFFSET2:
01AD 70                 MOV     M,B             ;PUT OFFSET2 IN TABLE
01AE 23                 INX     H               ;POINT OFFSET FOR COL 3
01AF 3D                 DCR     A               ;TEST FOR REMAINER OF 1
01B0 C2B401             JNZ     SETOFFSET3      ;IF REMAINDER <> 1, USE
                                                ;OFFSET3=OFFSET2
01B3 05                 DCR     B               ;ELSE OFFSET3=OFFSET2-1
                        SETOFFSET3:
01B4 70                 MOV     M,B             ;OFFSET TO COLUMN 4
                        ;
                        ;NOW THAT THE OFFSETS FOR THE COLUMNS HAVE BEEN DETERMINED
                        ;THE ACTUAL PRINT OUT CAN BE MADE.
                        ;
                        REPRINT:
01B5 F1                 POP     PSW             ;RECOVER FILE COUNT
                        REPRINT1:
01B6 F5                 PUSH    PSW             ;SAVE AGAIN FOR LATER USE
01B7 32F902             STA     FILE$COUNT      ;SAVE FOR COUNTING
01BA 3E0C               MVI     A,SCREEN$HGT    ;SET FOR VIDEO DISPLAY SIZE
01BC 32FB02             STA     LINE$COUNT
01BF 11A802             LXI     D,HEADING
01C2 0E09               MVI     C,9             ;BUFFER PRINT COMMAND
01C4 CD0500             CALL    BDOS            ;CP/M PRINTS HEADING
                        ;
01C7 21FB02             LXI     H,DIR$TABLE - 14 ;POINT DUMMY 0TH ENTRY
                        ;
                        PRINT$LINE:
01CA E5                 PUSH    H               ;SAVE BASE ADDRESS
01CB 11F302             LXI     D,OFFSET0       ;POINT OFFSET TABLE
01CE 3E04               MVI     A,NBR$COL       ;4 COLUMNS PER LINE
                        PRINT$NAME:
01D0 32FA02             STA     COLUMN$CNT      ;SAVE COUNT OF COLUMNS
01D3 E5                 PUSH    H               ;SAVE CURRENT NAME POINTER
01D4 D5                 PUSH    D               ;SAVE OFFSET TABLE POINTER
                        ;
01D5 11EA02             LXI     D,DOUBL$SPACE   ;PRINT 2 BLANKS
01D8 0E09               MVI     C,9             ;'PRINT BUFFER' COMMAND
01DA CD0500             CALL    BDOS            ;USE CP/M FACILITY
01DD D1                 POP     D               ;GET OFFSET TABLE POINTER
01DE E1                 POP     H               ;GET NAME POINTER
01DF 1A                 LDAX    D               ;GET OFFSET VALUE
                        ;
                        ;THE OFFSET VALUE IS THE NUMBER OF FILE NAMES FROM THE
                        ;CURRENT NAME POINTER TO MOVE THE PRINT POINTER.
                        ;
01E0 010E00             LXI     B,14            ;EACH NAME IS 14 LONG
                        MULT$14:
01E3 09                 DAD     B               ;ADD 14 FOR EACH OFFSET
01E4 3D                 DCR     A               ;UNTIL OFFSET = 0
01E5 C2E301             JNZ     MULT$14
01E8 E5                 PUSH    H               ;SAVE NEW NAME POINTER
01E9 D5                 PUSH    D               ;SAVE OFFSET POINTER
01EA EB                 XCHG                    ;POINT NAME TO PRINT W/<DE>
01EB 0E09               MVI     C,9             ;PRINT BUFFER
01ED CD0500             CALL    BDOS            ;PRINT THE FILE NAME AND
                                                ;ITS MENU NUMBER
                        ;
                        TEST$FINISH:
01F0 21F902             LXI     H,FILE$COUNT    ;SEE IF DONE PRINTING
01F3 35                 DCR     M               ;BY TESTING COUNT OF FILES
01F4 D1                 POP     D               ;GET OFFSET POINTER
01F5 E1                 POP     H               ;GET POINTER TO LAST NAME
```

```
01F6 CA0C02             JZ      FINISH          ;NO MORE TO PRINT.
01F9 13                 INX     D               ;ADVANCE OFFSET POINTER
01FA 3AFA02             LDA     COLUMN$CNT
01FD 3D                 DCR     A               ;SEE IF COLUMNS LEFT = 0
01FE C2D001             JNZ     PRINT$NAME      ;PRINT ANOTHER SAME LINE.
0201 CD9802             CALL    CRLF            ;MOVE TO NEXT LINE
0204 E1                 POP     H               ;GET BASE OF PREVIOUS LINE
0205 110E00             LXI     D,14            ;ADD OFFSET
0208 19                 DAD     D
0209 C3CA01             JMP     PRINT$LINE
;
;THE FILE NAMES AND THEIR MENU NUMBERS HAVE BEEN PRINTED.
;THIS NEXT LOOP OUTPUTS SUFFICIENT LINE FEEDS TO PUT THE
;HEADING AT THE TOP OF A 16 LINE VIDEO DISPLAY(THEREBY
;CLEARING THE SCREEN), AND PUTS THE REQUEST FOR USER
;SELECTION AT THE BOTTOM OF THE SCREEN.
;
;
FINISH:
020C E1                 POP     H               ;UNJUNK STACK
LF$LOOP:
020D CD9802             CALL    CRLF
0210 F20D02             JP      LF$LOOP         ;OMIT THIS LINE IF DESIRED
0213 11C402             LXI     D,PROMPT        ;POINT INSTRUCTION MSG.
0216 0E09               MVI     C,9
0218 CD0500             CALL    BDOS            ;AGAIN CP/M PRINTS MESSAGE
021B 11A706             LXI     D,INPUT$BUFF
021E 3E0A               MVI     A,10            ;TEN CHARACTERS MAX
0220 12                 STAX    D
0221 0E0A               MVI     C,10            ;'READ BUFFER' COMMAND
0223 CD0500             CALL    BDOS
;
;ON RETURN FROM BDOS LINE INPUT FUNCTION, THE DIGITS
;TYPED BY THE USER ARE IN THE BUFFER AT INPUT$BUFF+2.
;CONVERT TO BINARY.
;
0226 21A806             LXI     H,INPUT$BUFF+1  ;POINT CHAR COUNT
0229 7E                 MOV     A,M             ;GET IT AND SEE IF'> 2
022A FE03               CPI     3
022C D26501             JNC     REPRINT         ;REPRINT THE MENU
022F 4F                 MOV     C,A             ;COUNT OF DIGITS TO <C>
0230 0600               MVI     B,0             ;ZERO <B>
;
GET$MENU$NBR:
0232 23                 INX     H               ;POINT ASCII DIGIT
0233 7E                 MOV     A,M             ;GET IT
0234 CD8602             CALL    ASCII$CONVERT   ;CONVERT TO BINARY
0237 DA6501             JC      REPRINT         ;RE-DISPLAY ON ERROR
023A 0D                 DCR     C
023B C23202             JNZ     GET$MENU$NBR
;
;<B> HAS THE MENU NUMBER. TO BE SURE THIS IS A LEGAL MENU
;REQUEST, COMPARE WITH FILE$COUNT <STILL ON STACK).
;
023E F1                 POP     PSW             ;RECOVER FILE COUNT
023F B8                 CMP     B               ;FILE$COUNT-REQUEST NBR.
0240 DA8601             JC      REPRINT1        ;RE-DISPLAY MENU IF ILLEGAL.
;
;
0243 110E00             LXI     D,14            ;INCREMENT BETWEEN NAMES
0246 21F802             LXI     H,DIR$TABLE - 14 ;POINT DUMMY 0TH ENTRY
FIND$NAME:
0249 19                 DAD     D               ;ADD OFFSET <B> TIMES
024A 05                 DCR     B
024B C24902             JNZ     FIND$NAME
;
;AT THIS POINT <HL> POINTS THE SELECTED FILE NAME. NOW FIND
;THE ADDRESS OF CP/M SO THE PROPER COMMAND NAME AND THE
;SELECTED FILE NAME CAN BE PUT INTO THE COMMAND BUFFER.
;
024E EB                 XCHG                    ;SAVE POINTER TO FILE NAME
024F 2A0600             LHLD    6               ;GET BDOS ENTRY POINT
0252 01FAF7             LXI     B,- CCP$LEN     ;OFFSET TO START OF CP/M
0255 09                 DAD     B
0256 E5                 PUSH    H               ;SAVE CP/M ENTRY POINT
                                                ;ON STACK FOR BRANCH.
0257 010800             LXI     B,8             ;OFFSET TO COMMAND BUFFER
025A 09                 DAD     B               ;<HL> POINTS PLACE TO PUT
                                                ;NAME OF .COM FILE TO BE
                                                ;EXECUTED.
025B D5                 PUSH    D               ;SAVE POINTER TO FILE NAME
025C EB                 XCHG                    ;<DE> POINTS COMMAND BUFF
;
;SINCE THE SCAN POINTER IS NOT RESET BY REENTRY INTO CP/M
;THE SCAN POINTER MUST BE RESET BY THIS PROGRAM. THE SCAN
;POINTER IS STORED BY CP/M AT THE END OF THE COMMAND
;BUFFER.
;
025D 218000             LXI     H,128           ;OFFSET TO END OF CMD BUFF
                                                ;WHERE POINTER IS STORED
0260 19                 DAD     D               ;<HL> POINTS STORAGE PLACE
0261 73                 MOV     M,E             ;UPDATE BUFFER POINTER TO
0262 23                 INX     H               ;THE START OF THE COMMAND
0263 72                 MOV     M,D             ;BUFF SO CP/M WILL READ.
;
                        IF      BASIC$PROG      ;CONDITIONAL ASSEMBLY
                                                ;FOR BASIC TO RUN THE
                                                ;SELECTED PROGRAM
0264 21B802             LXI     H,COMMAND$NAME  ;POINT COMMAND NAME
0267 0E06               MVI     C,LEN$CMD$NAME  ;LENGTH OF COMMAND NAME
0269 CD7D02             CALL    BLOCK$MOVE
;
                        ENDIF
;
;
026C E1                 POP     H               ;POINT SELECTED FILE NAME
026D 0E08               MVI     C,8             ;LENGTH OF FILE NAME
026F CD7D02             CALL    BLOCK$MOVE
;
;*****************************************************
;IF THE VERSION OF BASIC TO BE USED DOES NOT REQUIRE THE
;FILE TYPE TO BE SPECIFIED IN THE COMMAND LINE, OMIT THE
;LINES BETWEEN THE ASTERICK LINES.
;
                        IF      BASIC$PROG      ;AGAIN FOR SOME VERSIONS
                                                ;OF BASIC WHICH REQUIRE
                                                ;THE FILE TYPE TO BE
                                                ;SPECIFIED.
0272 21C302             LXI     H,SPEC$TYPE     ;NEEDED BY SOME VERSIONS
                                                ;OF BASIC
0275 0E04               MVI     C,4             ;LENGTH OF 'TYPE' FIELD
0277 CD7D02             CALL    BLOCK$MOVE
;
                        ENDIF
;
;*****************************************************
;
```

```
                ;
027A AF         XRA     A               ;NEED A ZERO AT END
027B 12         STAX    D               ;OF COMMAND LINE
                ;
                ;THE ADDRESS OF CP/M IS ON THE STACK SO A SIMPLE RETURN
                ;WILL EXECUTE CP/M AND IN TURN EXECUTE `BASIC [FILENAME]'.
                ;
027C C9         RET
                ;
                ;SUBROUTINES:
                ;
                BLOCK$MOVE:
027D 7E         MOV     A,M
027E 12         STAX    D
027F 13         INX     D
0280 23         INX     H
0281 0D         DCR     C
0282 C27D02     JNZ     BLOCK$MOVE
0285 C9         RET
                ;
                ASCII$CONVERT:
0286 D630       SUI     '0'             ;SUBTRACT ASCII BIAS
0288 FE0A       CPI     9 + 1           ;BE SURE IT'S NUMERIC
028A 3F         CMC
028B D8         RC                      ;SET CY=1 IF ILLEGAL
028C 57         MOV     D,A             ;SAVE IN <D>
028D 78         MOV     A,B             ;GET PREVIOUS RESULT
028E 07         RLC                     ;MULTIPLY BY 2
028F 07         RLC                     ;THEN 4
0290 07         RLC                     ;THEN 8
0291 80         ADD     B               ;THEN 9
0292 D8         RC                      ;OUT OF BOUNDS
0293 80         ADD     B               ;THEN FINALLY BY 10
0294 D8         RC                      ;CY=1 ALWAYS MEANS ERROR
0295 82         ADD     D               ;ADD IN NEW RESULT
0296 47         MOV     B,A             ;SAVE IN <B>
0297 C9         RET
                ;
                CRLF:
0298 11A502     LXI     D,CRLFMSG
029B 0E09       MVI     C,9             ;BUFFER PRINT
029D CD0500     CALL    BDOS
02A0 21FB02     LXI     H,LINE$COUNT
02A3 35         DCR     M
02A4 C9         RET
                ;
02A5 0D0A24     CRLFMSG DB      0DH,0AH,'$'
                ;
02A8 0A0909092EHEADING DB       0AH,9,9,9,'.BAS  MENU',0DH,0AH,0AH,'$'
                ;
                        IF      BASIC$PROG      ;CONDITIONAL ASSEMBLY FOR
                                                ;BASIC.COM + FILENAME.
                ;
                COMMAND$NAME:
02BA 4241534943         DB      'BASIC '        ;INSERT THE NAME OF THE
                                                ;BASIC TO BE USED (E.G:
                                                ;`CRUN', `CRUN2', ETC.)
                ;
0006 =          LEN$CMD$NAME    EQU     $-COMMAND$NAME
                ;
                SPEC$TYPE:
02C0 2E42414(           DB      '.BAS'          ;ONLY IF REQUIRED BY BASIC
                                                ;VERSION USED.
                ;
                        ENDIF
                ;
                PROMPT:
02C4 454E544552         DB      'ENTER MENU NUMBER AND PRESS `RETURN'' $'
                ;
                DOUBL$SPACE:
02EA 202024             DB      '  $'
                ;
                MENU$BUFF:
02ED 202D202030         DB      ' - 0$'
                ;
                ;
02F3 01         OFFSET0 DB      1
02F4 000000     OFFSET1 DB      0,0,0
                ;
                END$OF$TABLE:
02F7 0903               DW      DIRTABLE
                ;
                FILE$COUNT:
02F9 00                 DB      0
                COLUMN$CNT:
02FA 04                 DB      4
                LINE$COUNT:
02FB 00                 DB      0
                ;
                ;IN THE NEXT LINE INSERT THE FILE TYPE OF THE FILES TO BE
                ;DISPLAYED.
                ;
                        IF      BASIC$PROG
                ;
                SRCH$FCB:
02FC 003F3F3F3F         DB      0,'????????BAS',0
                ;
                        ELSE                    ;FOR MACHINE LANGUAGE
                ;
                SRCH$FCB:
                        DB      0,'????????COM',0       ;FOR MACHINE
                                                        ;LANGUAGE GAMES
                ;
                        ENDIF
                ;
                ;
                DIR$TABLE:
0309 FF                 DB      -1              ;FORCE FIRST FILE
                                                ;TO BE PUT HERE
                ;
                STACK$AREA EQU  DIR$TABLE + 64*14 + 30
                ;
                INPUT$BUFF EQU  STACK$AREA
                ;
                ;EQUATES:
                ;
0005 =          BDOS    EQU     5
0004 =          NBR$COL EQU     4
0806 =          CCP$LEN EQU     3106H-2900H
0010 =          SCREEN$SIZE EQU 16              ;CHANGE AS APPROPRIATE
                                                ;FOR VIDEO
000C =          SCREEN$MGT  EQU SCREEN$SIZE - 4
                ;
```

Turn-Key, con't...

Microsoft
EDIT-80 Package

EDIT-80 is a random access, line oriented text editor similar to those used on large computers like the DEC PDP-10 and IBM 360. It may be used on any 8080 or Z-80 microcomputer system running the CP/M or TRSDOS Operating System. While it supports a full range of editing capabilities, EDIT-80 is still fast and easy to use. You will find it versatile enough to meet the most demanding text editing requirements.

Microsoft's MACRO-80 assembler and FORTRAN-80 compiler print listings and error messages with EDIT-80 line numbers, giving the user quick reference to source lines.

In addition to commands that insert, delete, print and replace lines of text, EDIT-80 offers the following features:

- **Alter Mode**

  Alter (or intraline) mode provides a full set of intraline subcommands to edit portions of individual lines. These subcommands give the user more extensive editing capabilities than those provided with the EDIT command in Microsoft BASIC.

- **Numbering**

  Use the Number command to renumber an entire file or just parts of a file. Handy when "making room" for an insertion or just organizing line numbers in a file.

- **Multiple-page Files**

  If desired EDIT-80 files may be divided into sections called "pages." Page divisions mean easy handling of large files (line numbers may be reused on different pages) or convenient markers for the logical subdivisions in a file.

- **Find and Substitute**

  Specified text is efficiently located or replaced with EDIT-80's global Find and Substitute commands.

- **File Parameters**

  EDIT-80 can be used to edit BASIC programs or files without EDIT-80 line numbers, and files may be output with or without line numbers.

Summary of EDIT-80 Commands

Alter	Enters Alter mode
Begin	Moves to the beginning of a page
Delete	Deletes lines
Exit	Writes text to disk and exits EDIT-80
Find	Finds text
Insert	Inserts lines
Kill	Deletes page marks
List	Prints lines at the line printer
Mark	Inserts a page mark
Number	Renumbers lines
Print	Prints lines
Quit	Exits the editor without writing text to disk
Replace	Replaces lines
Substitute	Finds and replaces text
Write	Writes text to disk
Extend	Allows extension of lines

FILCOM

The EDIT-80 package includes a file compare utility called FILCOM. FILCOM compares two files and outputs differences between them. Source files or binary files may be compared using FILCOM.

Prices

EDIT-80 PACKAGE including EDIT-80 Text Editor and FILCOM File Compare Utility supplied on single density, 8" CP/M diskette, with manual	$120.00
EDIT-80/FILCOM, manual only	$ 10.00

For more information, contact
Paul Allen
Microsoft
10800 NE Eighth, Suite 819
Bellevue, WA 98004
206-455-8080
Telex 328945

MICROSOFT